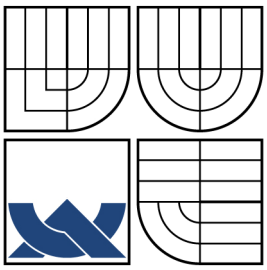
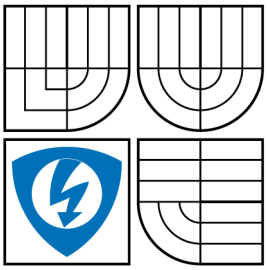


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

BEZPEČNÉ APLIKACE S MIKROKONTROLERY

SECURE APPLICATIONS WITH MICROCONTROLLERS

DIPLOMOVÁ PRÁCE

MASTER´S THESIS

AUTOR PRÁCE **Bc. JIŘÍ SOBOTKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MIROSLAV BALÍK, Ph.D**
SUPERVISOR

ANOTACE

Tato práce se zabývá problematikou bezpečného provozu mikrokontrolérů. Při dlouhodobém provozu mikrokontrolérů může dojít k jejich poruše či nesprávnému fungování. Tato práce si klade za úkol popsat jednotlivé druhy chyb, ke kterým v procesorech dochází, příčiny vzniku těchto chyb a možnosti předcházení poruch v mikrokontrolérech. Problémů provázejících provoz mikrokontrolérů je celá řada, a proto existuje i mnoho způsobů předcházení těmto problémům.

První část práce se zabývá popisem jednotlivých druhů chyb a pokusů o jejich modelování.

Druhá část této práce popisuje možnosti předcházení jednotlivých chyb během vývoje systému nebo i během provozu mikrokontroléru. Objasňuje způsoby tolerování chyb použitím redundantních obvodů v případě hardwarových chyb a úprav zdrojových kódů v případě softwarových chyb.

V třetí části práce jsou zkoumány metody analýzy rizik provázejících chod mikrokontrolérů. Rozpoznání událostí, které mohou způsobit poškození systému, je přiřazován velký význam. Analýza rizik probíhá během vývoje i během provozu systému.

Čtvrtá část se snaží popsat některé z technik testování procesorů, jako hlavní způsob nalezení chyb před zapojením do provozu.

V poslední části je pokus o praktický návrh testovacího algoritmu pro vybraný mikroprocesor a aplikaci jedné z analytických technik.

Klíčová slova: Tolerance chyb, pokrytí chyb, analýza rizika, analytická metoda, testování procesoru

ABSTRACT

This thesis deals with problematics of secure operations of microcontrollers. During long-term use, the microcontrollers can be affected by error or malfunction. The aim of this thesis is to describe the types of errors which can affect processors, cause of creation of these errors and possibilities of preventing malfunctions in microcontrollers. There is many problems connected with operation of microcontrollers, therefore there are many methods to prevent these problems.

First part of this thesis deals with description of types of mistakes and with attempts of theirs modelling.

Second part of this thesis narrate possibilities of preventing each type of mistake during designing the system or during operation of microcontroller and explaining methods of fault toleration by using redundant circuits in case of hardware fault and by modifying source code in case of software faults.

In third part the hazard analytical methods are described. The recognition of events able to damage the system has high importance. The risk analysis takes place during developing and during operation of the system.

Fourth part is trying to describe some of the processor testing techniques as main method of finding errors before operating use of the microcontroller.

Last part attempts to practical draft of testing algorithm for chosen microprocessor and application of one of analytical techniques.

Keywords: Fault toleration, fault coverage, risk analysis, analytical metod, procesor testing

Přehled použitých zkratek

ADC	Součet s Carry
ADD	Součet bez Carry
ADIW	Přičtení konstanty ke slovu
ALU	Aritmetic Logic Uni
AMBIST	Analog and Mixed-Signal Built-In Self-Test
AND	Logická funkce, provádí logický součin
ANDI	Logické AND s konstantou
ATPG	Automatic Test Pattern Generation
AVR	Rodina 8-bitových mikroprocesorů z produktů korporace Atmel
BIST	Built-In Self-Test
BRNE	Branch if not equal
C	Carry bit
CBR	Odstranění bitů z registru
CIA	Chemical Industrie Association
CLR	Vyčištění registru
CMOS	Complementary Metal–Oxide–Semiconductor
COM	Doplněk jedné
CP	Compare
CPC	Compare with carry
CPI	Compare with immediate
CRC	Cyclic Redundancy Check
DEC	Úbytek
EDVAC	Electronic Discrete Variable Automatic Computer
ETA	Event Tree Analysis
FAN	Fan-Out Oriented
FMEA	Failure Modes and Effects Analysys
FMECA	Failure Modes, Effect and Criticality Analysis
FMUL	Zlomkové násobení záporných
FMULS	Zlomkové násobení kladných
FMULSU	Zlomkové násobení kladných se zápornými
FTA	Fault Tree Analysis
HAZOP	Hazard and Operability Studies
HRA	Human Reliability Analysis
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INC	Přírůstek
K	Konstanta
LBIST	Logical Built-In Self-Test
LSI	Large Scale Integration
MBIST	Memory Built-In Self-Test
MOS	Metal–Oxide–Semiconductor
MUL	Násobení záporných
MULS	Násobení kladných

MULSU	Vynásobení kladných se zápornými
NEG	Doplněk dvou
NMR	N-Modular Redundancy
NP	Non Programmable
OR	Logická funkce, provádí logický součet
ORI	Logické OR s konstantou
PBIST	Programmable Built-In Self-Test
PES	Programmable Electronic System
PHA	Preliminary Hazard Analysys
PHI	Preliminary Hazard Identification
PHI	Preliminary Hazard Identification
PLC	Programmable Logic Controller
PODEM	Path-Oriented Decision Making
POST	Power-On Self-Test
RAM	Random Access Memory
Rd	Cílový (a zdrojový) registr v Souboru registrů
ROM	Read Only Memory
Rr	Zdrojový registr v Souboru registrů
SBC	Odečtení s Carry
SBC	Odečtení s Carry
SBCI	Odečtení konstanty s Carry
SBIW	Odečtení konstanty od slova
SBR	Nastavení bitů v registru
SER	Nastavení registru
SFMEA	Software Failure modes and Effect Analysis
SFTA	Software Fault Tree Analysis
SRA	Software Risk Analysis
SUB	Odečtení bez Carry
SUBI	Odečtení konstanty
TMR	Triple Modular Redundancy
TST	Test na nulu nebo na minus
TTL	Transistor-Transistor-Logic
USD	United States Dolar
VLSI	Very Large Scale Integration
XOR	Logická funkce Expendable OR

Obsah

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Radomíru Svobodovi, za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne …… ……..
(podpis autora)

1 Úvod

Počítačové systémy nám umožňují vykonávat úlohy, které nejsme schopni vykonat jinými způsoby – například provádění extrémně složitých výpočetních operací nebo rychlé hledání v obrovském množství informací. Ovšem také mohou klamat naše očekávání mnoha různými způsoby, někdy se zničujícími následky.

Tato práce se snaží popsat původ těchto selhání a způsoby jejich odvrácení. Popisuje možnosti zkoumání chybových stavů mikrokontrolérů, způsoby odstranění chyb v systémech nebo jejich tolerance. V následujících kapitolách jsou uvedeny způsoby analýzy chyb a chybových stavů, analytické metody zkoumající příčiny vzniku chybových stavů a druhy testování procesorů.

Výsledkem této práce je nastínění základních praktik jak zabránit selhání systémů řízených počítači nebo souvisejícími technologiemi a návrh algoritmu pro otestování aritmeticko-logické jednotky procesoru Atmel.

2 Pozadí vzniku rizika

Nejprve se budeme věnovat poruchovosti, hrozbám a rizikům v počítačích a souvisejících systémech.

- Poruchovost je slabina, která může vést k nežádoucím následkům.
- Hrozba představuje nebezpečí, že porucha může být záměrně využita nebo náhodně spuštěna.
- Riziko je možný problém, s příčinami a důsledky. Na riziko může být nahlíženo jako na poškození ve které může vyústit je-li hrozba uskutečněna, nebo jako na míru rozsahu poškození, jako výsledek pravděpodobnosti a výsledek následků. Explicitní vyjádření rizika je sama riskantnost provozu daného zařízení. Odvrácení rizika je nesmírně složitý úkol, který představuje všudypřítomný problém.

2.1 Terminologie

Následující termíny jsou aplikovány na počítačové a komunikační systémy, a jsou běžně zavedeny v technické praxi.

- Spolehlivost znamená, že systém funguje tak jak se od něho očekává, a to po celou dobu jeho činnosti. Spolehlivost je také míra toho, jak systém splňuje předpoklady v průběhu jeho činnosti, za specifikovaných stavů prostředí. Hardwarová spolehlivost určuje schopnost systému odolávat selhání. Softwarová spolehlivost určuje schopnost systému vykonávat určité provozní požadavky.
- Bezpečnost zahrnuje zbavení se nebezpečí, nebo více specificky, zbavení se nepřijatelných událostí, zákeřného a náhodného zneužití. Bezpečnost je také schopnost systému odolávat průniku zvenčí a zneužití zevnitř.
- Integrita znamená, že určité žádoucí podmínky jsou udržovány během celého provozu. Například, integrita systému souvisí s rozsahem hardwarových a softwarových změn, které byli na systému nevhodně provedeny. Integrita dat popisuje změny dat, neboli jaká data jsou a jaká by měla být. Integrita osob popisuje individuální chování v požadovaném návyku.

Tyto tři termíny se zdánlivě významně překrývají, a skutečně tomu tak je, nezávisle na tom, jak důkladně jsou definice jednotlivých termínů zvoleny. Tyto koncepty jsou z podstaty navzájem propojeny, použité technologie je musí nutně pokládat za společné. Kromě toho, nikdy si nemůžeme být absolutně jisti kdy je systém bezpečný nebo spolehlivý.

2.2 Perspektivy

Náš současný styl života je se stoupající měrou závislý na počítačových systémech pracujících v aplikacích s extrémními nároky na bezpečnost, jejichž selhání může vyústit ve vážné následky. Extrémní nároky na bezpečnost zahrnují ochranu lidských životů a zdrojů, na nichž závisí naše bytí, při zajištění adekvátní spolehlivosti systému, ochrany dat a včasné reakce, obzvláště pod zvýšeným rizikem. Snahou mnoha návrhů počítačových systémů je vytvoření aplikace reagující jako celek, schopné současně plnit množství bezpečnostních požadavků, a musí toho být schopná nejen během provozu ale i během údržby a i z hlediska dlouhodobého vývoje. Splnění každého požadavku je dost složité, ale současné a nepřetržité plnění rozdílných a někdy i konfliktních požadavků je většinou mnohem více složité.

V minulosti došlo k mnoha různým selháním počítačových systémů. Pochopení příčin těchto selhání nám může pomoci přinejmenším ke snížení pravděpodobnosti vzniku stejné chyby v budoucnu. Nicméně k těmto chybám stále dochází a nadto jsou stále veliké mezery mezi teorií a praxí, a mezi výzkumem a vývojem.

Při návrhu systému můžeme narazit na mnoho záludností ve snaze splnit bezpečnostní požadavky. Techniky návrhů softwaru představují řešení, ale bez záruk. Zkušenosti nám ukazují, že i ty nejdůsledněji navržené systémy mohou obsahovat závažné chyby. Testování i formální ověřování má vážné nedostatky, například přirozená nedokonalost prvního a značná péče a úsilí potřebné k provedení druhého. Celkově vzato, nejsou zde žádné snadné odpovědi.

Bohužel i v případě, že ideální systém byl navržen tak, že jeho specifikace byla shledána v souladu s jeho bezpečnostními požadavky, a návrh byl korektně realizován takže kód může být shledán v souladu se specifikací, systém samotný stále nemůže být absolutně důvěryhodný. Požadované chování může být podryto selháním základních předpokladů (ať už implicitních nebo explicitních) byť i jen přechodným. Takovéto předpoklady jsou dalekosáhlé, i když občas nejsou dokonce ani stanovené. Například, když jsou požadavky kompletní a správné, takže v návrhu nejsou žádné skryté chyby, žádný škodlivý kód nebyl vložen, nejsou zde žádní škodlivý uživatelé (z vnitřku nebo z vnějšku), ani data a ani systém nebyli nesprávně změněny, a hardware funguje dost spolehlivě a předvídatelně takže ochrana proti selhání je dostatečná. Ovšem lidské zneužití nebo jiné nepředpokládané problémy mohou rozvrátit i ty nejpečlivěji navržené systémy.

A tak zde jsou dobré zprávy a také špatné zprávy. Dobré zprávy jsou že počítačové systémy se stále vyvíjejí. Když jsou dány dobře definované a rozumně náročné požadavky, talentovaní a pilní lidé, osvícený a rozumný management, adekvátní finanční a fyzické zdroje a přiměřeně spolehlivý hardware, systémy mohou být vybudovány tak, že ve většině případů jsou dostatečně schopné plnit dané striktní požadavky. Došlo zde ke značným pokrokům, zejména v oblasti výzkumu a v technologiích pro stavbu takového počítačového systému. Špatné zprávy jsou že garantovaného chování systému je nemožné dosáhnout s lidmi nebo bez lidí v provozním cyklu. Vždy zde mohou být okolnosti které nikdo není schopný ovlivnit, jako například záplavy, zásah blesku a kosmické záření. Kromě toho, lidé jsou chybující. A tak je zde vždy podstatné riziko při spoléhání se na počítačové systémy pracující s bezpečnostním rizikem – zvláště u těch, které jsou složité a nezbytné pro řízení aplikací v reálném čase, například letadlo v systému letecké kontroly, které je aerodynamicky nestabilní a nedokáže letět bez aktivního počítačového řízení. I to nejúplnější (ale stále ne úplné) testování zanechává pochybnosti. Mimoto mají vývojáři tendenci být pomalí

v přijímání nových poznatků a užitečných vývojových konceptů, a v zamítání mnoha dalších nápadů, které praktické nejsou. A co je více důležité, podstatné ve vývojářském úsilí a v provozu systému, že všechny potenciální katastrofy nemohou být předpovězeny, a většinou jsou to ty nepředpovězené okolnosti, které jsou nejvíce zničující, typicky kvůli kombinaci okolností zahrnujících jak lidi tak počítače. Zásadním závěrem je to, že i když jsme extrémně opatrní a máme štěstí, stále musíme předvídat výskyt vážných katastrof při použití počítačových systémů v kritických aplikacích.

Nikdy nesmíme předpokládat neomylnost technologie i lidí vyvíjejících a aplikujících tuto technologii. Samozřejmě v některých situacích může být pro nás příliš vysoké riziko spolehnout se na počítače nebo lidi, proto může být lepší nepoužívat přednostně automatické řízení procesu. V jiných aplikacích může být vhodná péče ve vývoji systému a v provozu dostačující k udržení možného rizika v akceptovatelném rozsahu. Ale všechny tyto řešení závisí na přesném ohodnocení rizik a jejich následků, na ohodnocení, které je většinou nedostatečné.

2.3 Zdroje rizika

V této kapitole jsou popsány některé z mnoha úseků vývoje a použití systému, během kterých může dojít ke vzniku rizika.

2.3.1 Zdroje vzniku problémů ve vývoji systému

Problémy mohou nastat během kteréhokoliv z úseků vývoje systému (často díky lidskému faktoru jako základní příčina), zahrnujíc následující:

- *Koncept systému* – například nevhodně použitá technologie, s příliš velikou pravděpodobností vzniku rizika, nebo nepoužití automatizace, kde by to bylo přínosem
- *Definice požadavků* – chybné, neúplné nebo rozporné požadavky
- *Design systému* – chybné pojetí nebo vady v návrhu nebo ve specifikaci ať už hardwarové nebo softwarové části
- *Hardwarová a softwarová implementace* – chyby při výrobě čipu nebo v zapojení, programové chyby, náhodně nebo záměrně použité škodlivé části kódu, schopné spuštění nepředpokládaných efektů (např. Trojský kůň nebo virus)
- *Podpůrný systém* – nedostačující programovací jazyk, vadné kompilátory a debuggery, nesprávné vývojové prostředky, chyby vzniklé při úpravě
- *Analýza koncepce a designu systému* – analýzy založené na mylných fyzikálních předpokladech, operační prostředí, lidské chování – chybné modely, chybné simulace
- *Analýza implementace* – například pomocí testování nebo verifikace kódu – nedokončené testování, chybná verifikace, chyby při odstraňování chyb
- *Vývoj* – ledabylá přestavba a údržba, špatně pochopená modernizace systému, vznik nových chyb při snaze o odstranění starých, rozšiřování systému do přílišné složitosti (“poslední stéblo trávy které zlomí velbloudí záda“)
- *Vyřazení z provozu* – předčasné odstranění primárního nebo záložního zařízení před plným zprovozněním nového systému, skrytá závislost na staré verzi, která již není k dispozici, ale její přítomnost je stále nezbytná (např. z důvodů kompatibility); neschopnost vyřadit systém před tím než jeho provoz začne být nekontrolovatelný (efekt albatros)

2.3.2 Zdroje vzniku problémů během provozu a použití systému

Problémy se mohou také vyskytnout během provozu a použití systému (většinou je viníkem lidský faktor nebo zásah zvenčí) zahrnujíc následující:

- *Vliv přirozeného prostředí* – osvětlení, zemětřesení, záplavy, extrémní teploty (jako v případě katastrofy raketoplánu Challenger), elektromagnetické a jiné rušení včetně kosmického záření a vlivu slunečních skvrn a mnoho dalších přirozených událostí
- *Zvířata* – žraloci, veverky, opice, myši, ptáci a další druhy mohou způsobit poškození částí nebo celého systému
- *Vliv infrastruktury* – ztráta elektřiny nebo klimatizace
- *Selhání hardwaru* – selhání zařízení z důvodů stárnutí materiálu, přechodného chování nebo všeobecně z přirozených důvodů – viz předchozí
- *Nesprávné chování softwaru* – neočekávané chování vyplývající z nezjištěných problémů v procesu vývoje systému nebo z pozdějších změn v systému, nebo ze špatné údržby po instalaci softwaru
- *Selhání při komunikaci* – výpadky přenosu, přirozené i úmyslné rušení
- *Vliv lidského faktoru při používání systému* – systémový operátoři, administrátoři, zaměstnanci, uživatelé nebo náhodní kolemjdoucí mohou způsobit problémy během celé doby provozu systému, například:

    - Instalace: Nesprávná konfigurace, chybné zavádění, nekompatibilní verze, chybné nastavení parametrů, chyby ve spojení

    - Nesprávné požití celého zařízení nebo počítačového systému: Mezi tyto problémy patří záměrné zneužití jako třeba zadání nesprávných dat, nedodržování časového harmonogramu, nesprávné pochopení výstupních dat, nereagování na výstražné signály, provádění špatných funkcí, což zahrnuje i úmyslné zneužití povolanými i nepovolanými osobami, zlomyslné vložení Trojského koně, podvod, a podobně

2.4 Nepříznivé efekty

Počítače ovlivňují náš život v mnoha oblastech, ve kterých je třeba být na vznik rizika připraven. Několik z těchto oblastí je zde popsáno spolu s druhy rizik které mohou v těchto oblastech vzniknout.

- Počítače a komunikace – ztráta nebo poškození komunikačního média, vkládání falešných nebo humorných zpráv nebo příkazů, ztráta soukromí odposlechem vedení nebo počítačů
- Počítače použité v kosmických aplikacích – ztráta životů, selhání misí, odklady startů, nezdařené experimenty, velké finanční překážky
- Počítače použité v obraně a ve válečných konfliktech – záměna přítele za nepřítele, selhání při útoku, selhání při obraně, neúmyslný útok, neúmyslná sebedestrukce, střelba do vlastních řad

- Počítače použité v dopravě – úmrtí, zpoždění, požáry, náhlé zrychlení, neschopnost zastavit nebo ovládat vozidlo, neschopnost opustit automaticky ovládané vozidlo po výpadku proudu
- Počítače použité pro kontrolu životně důležitých zařízení – úmrtí, zranění, zpoždění, nepraktičnost
- Počítače a příbuzné systémy použité ve zdravotnictví a v ochraně lidských životů – úmrtí, zranění, psychické poškození, špatná diagnóza, špatná léčba, špatné vyúčtování
- Počítače a příbuzné systémy, jejichž použití způsobuje zdravotní problémy – fyzické zranění, stres, psychická muka
- Počítače použité ve výrobě elektrické energie – úmrtí, zranění, výpadky proudu, dlouhotrvající ohrožení života zahrnujíc zamoření radiací v jaderných systémech
- Počítače použité k řízení toku peněz – podvod, porušení soukromí, ztráty a narušení obchodních zařízení jako například zavření burz a bank – náhodně i úmyslně
- Počítače, které dohlížejí na průběh voleb – náhodné špatné výsledky a volební podvody za použití počítačů
- Počítače, které dohlížejí nad provozem vězení – útěky za pomoci techniky, chyby vedoucí k propuštění zločince, selhání počítačem ovládaných zámků
- Počítače použité při výkonu spravedlnosti – zatčení a uvěznění nesprávných osob, neschopnost zatknout podezřelého
- Počítače použité k ochraně systémů a integrity dat – vnik Trojských koňů a virů, odmítnutí přístupu, falešné útoky, převraty
- Počítače sloužící k ochraně soukromých dat – narušení soukromí jako nechtěný přístup k datům nebo sledování činnosti uživatelů
- Počítače při všeobecném použití – náhodné špatné výsledky, zpoždění a ztráta dat, pronásledování úmyslným zneužitím systému pomocí škodlivého software a falešnými zprávami
- Počítače, které jinak nepříznivě ovlivňují lidské životy – osoby mohou být mylně prohlášeny za mrtvé, vymáhány neexistující poplatky, chyby v určování osob

2.5 Zranitelnost zabezpečení počítačových systémů

V neformální definici bezpečnost znamená osvobození od nepřijatelných událostí, zahrnujíce zákeřné a náhodné zneužití. V počítačovém smyslu nepřijatelné události mohou také zahrnovat následky selhání hardwaru. V běžné technické praxi se všeobecně termíny počítačová bezpečnost a komunikační bezpečnost označuje ochrana proti zneužití lidmi, vylučuje se ochrana proti selhání.

Nejpoužívanější termíny v bezpečnosti systémů jsou:

- *Důvěrnost, utajenost* – znamená, že informace je chráněna proti nezáměrnému prozrazení. Mechanismy a zásady počítačových systémů si vynucují utajení – například pro ochranu práv jednotlivce na soukromí nebo pro účely národní obrany. Systém neumožňuje náhodné prozrazení citlivých informací, a je odolný proti pokusům o získání přístupu k takovýmto informacím.
- *Integrita* – doslovně znamená, že prostředky jsou udržovány v nezměněných podmínkách, nebo je zde přesné dodržování specifikací (daných požadavků nebo hodnot). V počítačovém smyslu, *integrita systému* znamená že systém a systémová

data jsou udržovány v dostatečně správném a souvislém stavu, kdežto *integrita dat* znamená že data jsou také udržována. V tomto smyslu nebezpečí ztráty integrity je čistě starost bezpečnosti. Integrita může souviset s nezáměrným prozrazením, ale všeobecně je na ni nahlíženo více jako ochrana proti nezáměrné modifikaci. Uvnitř systému integrita zahrnuje zajištění *vnitřní konzistence*. Z celkového pohledu také integrita systému zahrnuje *vnější konsistenci*. To znamená, že vnitřní zařízení systému přesně odrážejí vnější svět. Související s vnější konsistencí jsou pojmy jako legálnost prvků systému a kompatibilita s vnějšími standardy. Integrita v širším smyslu také zahrnuje lidské úsudky.

- *Dostupnost* – znamená, že systémy, data a další prostředky jsou připraveny k použití v momentě, kdy jsou potřeba, navzdory výpadkům subsystémů a narušení pracovního prostředí. Z hlediska spolehlivosti je dostupnost všeobecně zvýšena pomocí konstruktivního navýšení prostředků, včetně alternativních přístupů a různých záložních mechanismů. Z hlediska bezpečnosti je dostupnost zvýšena pomocí opatření vedoucím k zabránění úmyslného odmítnutí služby.
- *Aktuálnost* – je důležitá zejména v real-time systémech, které musí uspokojit určité neodkladné požadavky, a je zde jistota, že potřebné prostředky jsou dostatečně rychle dostupné když jsou potřeba.

A znovu, je zde značné překrytí mezi těmito různými termíny. Důležité jsou také požadavky na lidskou bezpečnost a na celkovou životnost aplikace, obojí závisí na splnění požadavků souvisejících se spolehlivostí a bezpečností:

- *Spolehlivost systému* – je to schopnost systému na udržení dostupnosti prostředků, navzdory nepříznivým okolnostem jako například selhání hardwaru, softwarovým chybám, zákeřným aktivitám uživatele a nebezpečí z pracovního prostředí jako třeba elektrické rušení.
- *Lidská bezpečnost* – označuje ochranu jak jednotlivců, tak i skupin lidí. V současném kontextu se vztahuje k bezpečnosti kohokoliv, kdo je nějakým způsobem závislý na uspokojivém chování systému a na náležitém použití technologie. Lidská bezpečnost s respektem k celé aplikaci může záviset na spolehlivosti a bezpečnosti systému.

Opět zde je jasné překrytí mezi významem těchto termínů. Ve smyslu, kdy všechna zahrnují ochranu před nebezpečím, jsou všechny součásti ochrany. Ve smyslu, kdy ochrana proti nebezpečí zahrnuje údržbu některých konkrétních podmínek, mohou být uvažovány jako součásti integrity. Ve smyslu, kdy by se systém měl za všech podmínek chovat předvídatelně, všechny jsou součásti spolehlivosti. Různé otázky spolehlivosti jsou také zcela jasně zahrnuty jako součásti bezpečnosti, ačkoli mnoho dalších otázek spolehlivosti je obvykle posuzováno nezávisle. Nicméně nakonec je bezpečnost, integrita a spolehlivost vzájemně blízce související. Jakýkoliv, nebo všechny z těchto konceptů můžou být součástí požadavků v konkrétní aplikaci, a selhání zachování jakéhokoliv z nich může kompromitovat schopnost zachovat zbývající. Proto nehledáme jasné rozdíly mezi těmito termíny, preferujeme jejich použití více či méně intuitivně.

3 Tolerance chyb

Objektivní použití tolerance chyb znamená navrhnout systém takovým způsobem, kdy možné chyby nevyústí v selhání celého systému.Všechny metody tolerance chyb jsou založeny na nějaké formě **redundance**. To znamená, že máme systém, který je složitější než je třeba pro vykonání požadované úlohy. Toto zvýšení složitosti může mít více provedení, a v této práci se pokusím nastínit několik metod implementace redundance.

Tolerance chyb není nový pojem. První počítače, jako třeba EDVAC vyrobený v roce 1949, používaly duplikované aritmeticko logické jednotky (ALU) pro detekci chyb ve výpočtech. Velkých pokroků bylo dosaženo v padesátých letech, kdy je většina teoretického výzkumu přisuzována von Neumannovi (1956)[2]. V těchto ranných dobách byla tolerance chyb široce použita ke zvládnutí selhání v relativně nespolehlivém hardwaru, který byl tenkrát založen na principu elektronky. Jak se počítačová průmysl vyvíjel, hardware začal být spolehlivější, ale taky složitější. V moderních počítačových systémech je většina této složitosti implementována v softwaru.Aby byla tolerance chyb efektivní, musí poskytnout ochranu před chybami v návrhu, v softwaru a stejně tak proti hardwarovým chybám. Dnes je tolerance chyb využívána jako základ téměř všech bezpečnostních systémů, poskytujíc ochranu proti širokému spektru chyb. Také je zeširoka použita ve složitých součástkách, jako jsou paměťové moduly a mikroprocesory.

Motivace pro použití chybové tolerance se může zřetelně lišit mezi jednotlivými aplikacemi. Charakteristiky mohou být vylepšeny použitím technik pro zvýšení spolehlivosti, dostupnosti a bezpečnosti. Tyto charakteristiky jsou stále ve vzájemném vztahu, ale ve specifických aplikacích často jedna vyžaduje převahu. Tolerance chyb může být například použita k dosažení vysoké životnosti podmořského telefonního opakovače, k dosažení vysoké dostupnosti bankovního systému, ke zvýšení pravděpodobnosti úspěchu mise při letu raketoplánu nebo ke snížení nebo odložení údržby komunikačního satelitu.

V této seminární práci se především zaměřuji na bezpečnost, ačkoli jsou tyto problémy těsně spojeny otázkami spolehlivosti. Za určitých okolností se mohou požadavky na bezpečnost lišit od snahy o dosažení co nejvyšší spolehlivosti. V některých systémech může být bezpečnosti dosaženo na základě detekce chyb a některých mechanismů schopných přinutit systém k selhání do bezpečného stavu. S metodami účinné detekce chyb jako opatření před selháním může být systém bezpečný, i když obsahuje mnoho chyb a předtím byl nespolehlivý.Nicméně ve všech praktických systémech má spolehlivost vysokou důležitost, a jakýkoliv systém který je nespolehlivý je se vší pravděpodobností neúspěšný, i když je bezpečný. Tolerance chyb může být použita k zabránění selhání systému působením systémových chyb, a proto může být použita ke zvýšení spolehlivosti a dostupnosti systému. Zvýšená spolehlivost je vždy prospěšná a v mnoha případech je hlavním záležitostí ve zvyšování bezpečnosti systému.

Důležitým faktorem ovlivňujícím návrh každého systému s tolerancí chyb je zjištění počtu možných selhání systému, které mohou být akceptovány v daném časovém okamžiku. Analýza rizika se používá k zadání takové bezpečnostní úrovně do systému, která v tomto ohledu vyjadřuje jeho požadovanou výkonnost. Stupeň selhání dosažený systémem pomůže návrhářům ve výběru architektury systému. Během procesu vývoje je nezbytné odhadnout pravděpodobnost selhání navrhovaného systému, pro ověření, že splní svoje požadavky.
V následující kapitole se podíváme na různé techniky, které jsou použity k dosažení tolerance chyb a v další kapitole se pokusíme analyzovat provedení a spolehlivost těchto technik.

3.1 *Druhy chyb*

Chyby mohou být charakterizovány mnoha způsoby. Například mohou být charakterizovány podle jejich povahy, trvání nebo závažnosti.

3.1.1 Povaha chyb

Povaha chyb souvisí s jejich typem. Chyby mohou být charakterizovány jako náhodné nebo systematické. Prvořadou příčinou náhodných chyb je selhání hardwarové součásti. Všechny reálné komponenty mají omezenou příležitost k selhání v daném časovém úseku, a je tedy možné sestavit statistiku pro podobná zařízení která může být použita k odhadu výskytu chyb. Když jsou jednoduché součásti sestaveny do modulů, metody použité k sestavení komponent a techniky použité k jejich propojení můžou také způsobit chyby, a i zde nám zkušenosti umožňují odhadnout výskyt takových chyb. Protože jejich náhodná povaha neumožňuje předpovědět kdy konkrétní součástka nebo modul selže, pomůže nám statistická analýza odhadnout pravděpodobnost selhání v dané časové periodě.

Kromě náhodných hardwarových selhání, jsou systémy také předmětem **systematických chyb** tří hlavních typů – chyby ve specifikaci systému, chyby v softwaru a chyby v návrhu hardwaru. Běžně se tyto chyby posuzují jako rozdílné formy **chyb v návrhu**. Systematické chyby se hůře analyzují než náhodné chyby a je všeobecně nemožné předpovědět jejich vliv na chování systému. Mnoho zkoumání bylo zaměřeno na problém předpovídání pravděpodobnosti vlivu softwarových chyb na výkon systému.

Může být zřejmé, že k hardwarovým chybám dochází jak díky náhodným selhání komponent tak i díky chybám v návrhu. Neexistuje něco takového jako je náhodná softwarová chyba.

3.1.2 Trvání chyby

Chyby mohou být také posuzovány podle doby jejich trvání. Chyby, které se vyskytují po neurčitě dlouhou dobu nebo dokud nejsou odstraněny nějakým zásahem, jsou nazývány **permanentními chybami**. Chyby v návrhu, včetně softwarových chyb, jsou vždy permanentní, stejně jako mnoho chyb v hardwarových součástech. Některé chyby se mohou objevit a pak zase zmizet v krátkém čase. Tyto chyby jsou nazývány **přechodné chyby**. Příkladem přechodné chyby může být vliv alfa částice, která zasáhne polovodičový paměťový čip. K těmto událostem dochází velmi zřídka, ale mohou změnit stav jednoho nebo více bitů v paměti bez způsobení trvalého poškození zařízení. Ačkoliv je selhání přechodné, chyba v systému kterou způsobí zůstane i poté, kdy selhání samo odezní a musejí být podniknuty kroky k opravení této chyby jestliže chceme zabránit selhání celého systému. Třetím druhem chyb patřící do této skupiny jsou **střídavé chyby**. Tyto chyby se objeví, poté zmizí a pak se zase později objeví. Takovéto chyby mohou vzniknout ze špatně spájeného spoje, nebo díky korozi na kontaktech, takže ke správnému spojení dochází jenom někdy. Střídavé chyby mohou být také způsobeny vlivem elektrického rušení a jiným problémů s elektromagnetickou kompatibilitou. Díky jejich povaze je někdy velice obtížné střídavé chyby dekovat a odstranit, protože proces detekce chyby musí probíhat současně s existencí chyby. Naneštěstí se mnoho permanentních chyb jeví jako střídavé, protože jejich působení je patrné jenom v určitou dobu. Například chyba v synchronizaci softwaru je permanentní, protože její kód je vždy přítomen, ale její vykonání vede jenom někdy k selhání, v závislosti na načasování. Takové chyby, které mají charakteristiku střídavých chyb, je rovněž velmi těžké lokalizovat.

Selhání hardwarových součástí může být permanentní, přechodné nebo střídavé. Nicméně by mělo pamatováno na to, že chyby z těchto skupin mohou mít za následek chyby celého systému, které v něm setrvávají.

3.1.3 Závažnost chyby

Další klasifikace chyb podle jejich závažnosti je rozdělena na části systému, ve kterých mohou nastat. Lokalizované chyby mohou postihnout pouze jediný hardwarový nebo softwarový modul, kdežto globální chyby mají vliv na chování celého systému.

3.1 Hardwarové chyby

Jak jsme zjistili, hardwarové chyby mohou být způsobeny náhodným selháním hardwarové součásti nebo návrhovou chybou. Mohou být také permanentní, přechodné nebo střídavé, a mohou mít lokální nebo globální vliv.

Bez ohledu na povahu chyby je často užitečné vytvořit model jejího vlivu na chování systému. Toho může být dosaženo použitím jednoho z mnoha **chybových modelů**. Tak jako modely použité v jiných oblastech inženýrství, nejsou tyto modely vždy perfektním znázorněním fyzikálního vlivu, jenž se snaží popsat. Nicméně jejich použití značně usnadňuje analýzu složitého systému a může přispět při návrhu testovacích procedur a při simulaci okolností selhání.

3.2.1 Chybové modely

Provoz elektrického obvodu může být posuzován v mnoha úrovních. Na **úrovni atomu** sledujeme jednotlivé součástky jako jsou rezistory a tranzistory, a jejich propojení. Na této základní úrovni může být většina chyb viděna jako nedostatečně spojené obvody, nebo obvody nesprávně připojené na jiné vedení. V praxi mohou nastat různé okolnosti přerušení vodiče, k čemuž může dojít fyzickým přerušením elektrického vedení nebo nedokonalým spojením v součástce, jako je rezistor nebo tranzistor. Dále může dojít k nesprávnému propojení různých uzlů obvodu. K tomu může dojít mezi vnitřními signálovými cestami a nebo mezi signálovou cestou a jednou z napájecích linek. Tyto okolnosti mohou vyjadřovat fyzické zkraty nebo přemostění mezi sousedním vedením v obvodu, nebo vnitřní zkraty uvnitř součástek. Samozřejmě nemohou být všechny okolnosti selhání vyjádřeny použitím těchto jednoduchých modelů, ale v praxi velké procento skutečných selhání může být adekvátně vyjádřeno pomocí jednoho nebo více těchto modelů.

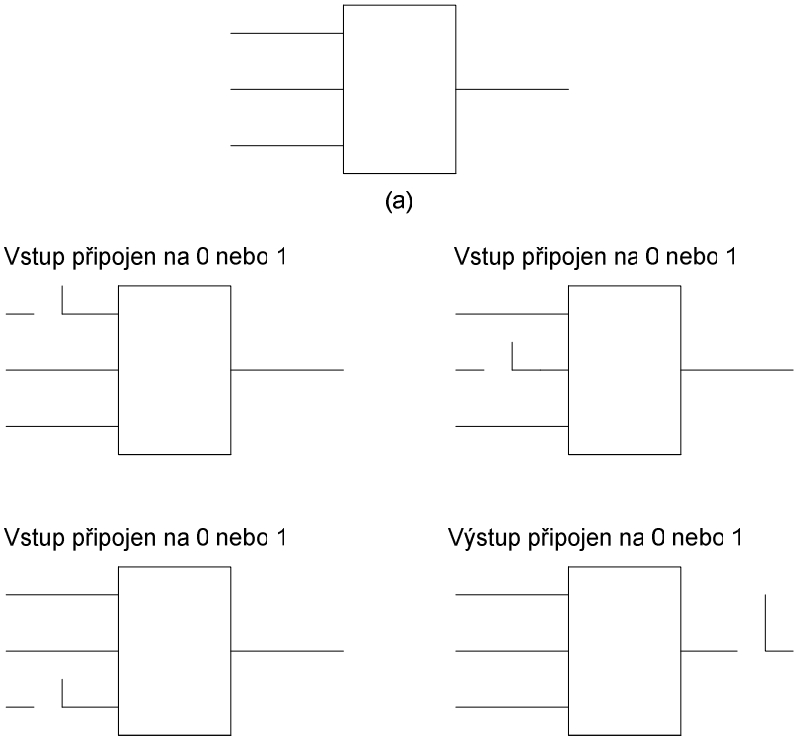
Ačkoliv model na atomové úrovni popsaný výše poskytuje dobrý obraz vlivu chyby na chování obvodu, je zřejmé, že posuzování každé součásti vede k značné složitosti. Jedna z metod na snížení této složitosti v digitálním obvodu je dívat se na systém spíše na modulární úrovni než na atomové úrovni.

Několik chybových modelů je použito k vyjádření rozdílných druh chyb. Zde budeme uvažovat tři typy:

- model trvání jedné chyby
- model zkratu způsobeného chybou
- model neurčité chyby

3.2.1.1 Model trvání jedné chyby

Jeden z prvních a taky jeden z nejvíce používaných modelů, je model trvání jedné chyby. Tento model se používá od padesátých let (Eldred, 1959; Kohavi, 1978) [2]. Tento model se nesnaží modelovat nitřní strukturu modulu, ale jednoduše uvádí, že selhávající modul může být charakterizován pomocí jeho vnějšího chování. Model předpokládá že chyba v modulu způsobí, že jeden ze vstupů nebo výstupů se zasekne na logické 1 nebo na logické 0. Také předpokládá že základní funkce obvodu jinak neovlivněné, a že chyba je permanentní. Model trvání jedné chyby nemůže přesně vyjádřit všechny okolnosti vzniku chyby. Samozřejmě nemůže vyjadřovat přechodnou nebo střídavou chybu, nicméně to dovoluje modelovat velké množství chyb jednoduchým způsobem. Ve skutečnosti většina chyb vzniklým díky přerušeným vedením, nepřipojeným nebo zkratovaným součástkám a zkratům mezi vedením může být reprezentována modelem trvání jedné chyby.

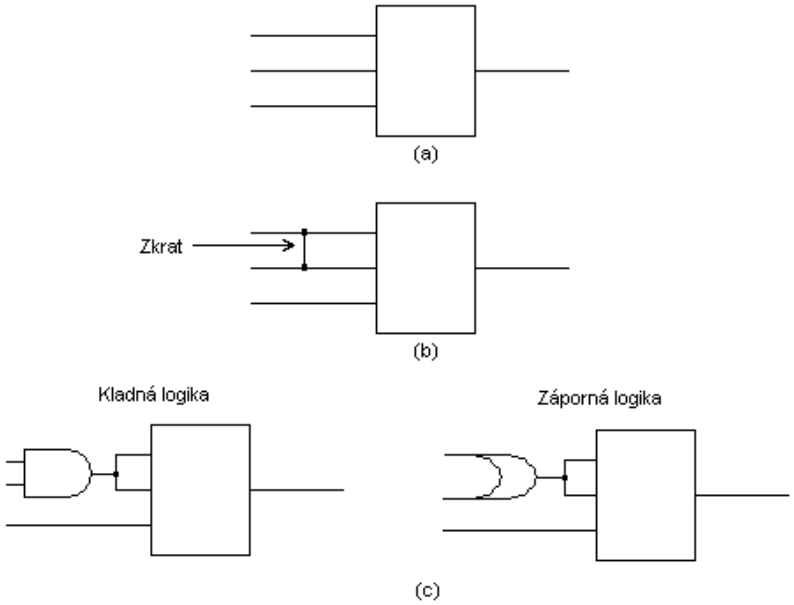


Obr. 1: Příklady trvání jedné chyby: (a) modul bez chyby, (b)možné trvání jedné chyby

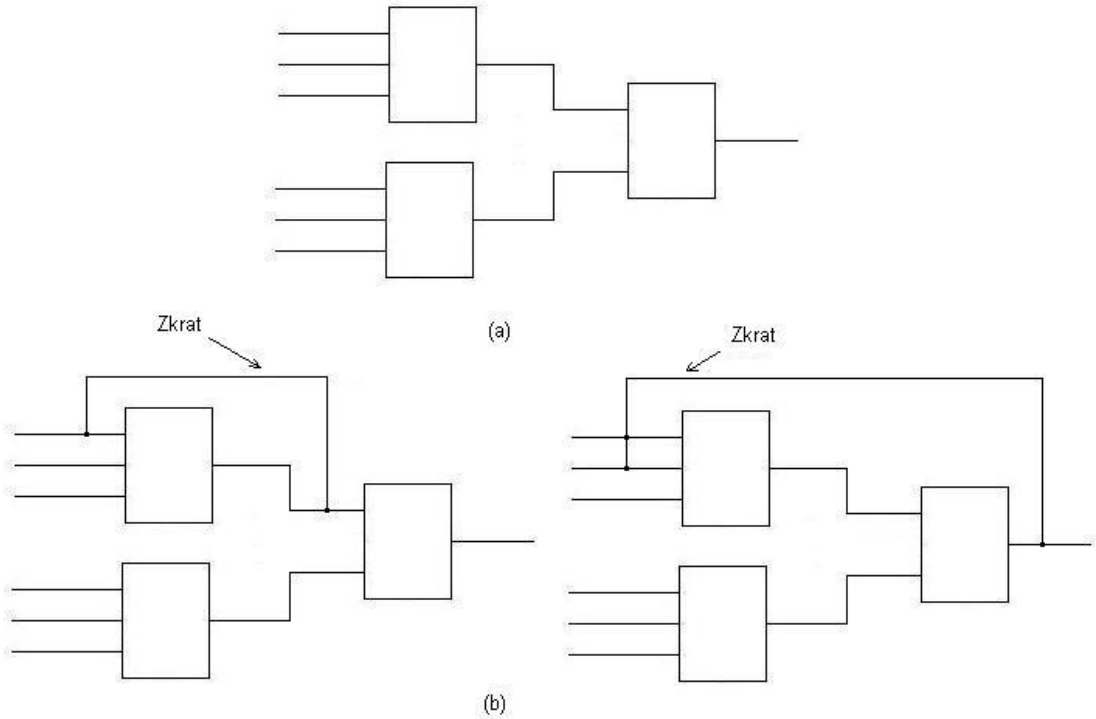
3.2.1.2 Model zkratu způsobeného chybou

Zkrat díky chybě nastane, když jsou jeden nebo více uzlů v obvodu náhodně spojeny dohromady, čímž vznikne permanentní chyba. Ačkoliv taková chyba může být občas vyjádřena modelem trvání jedné chyby, není to vždy možné. Použitím kladné logiky, vliv takovéto chyby je srovnatelný se spojením zúčastněného vedení pomocí operátoru AND. Použitím negativní logiky je tento vliv srovnatelný s OR operací. Výsledkem je nezáměrná logická operace odpovídající funkci AND nebo OR. Velmi jednoduchým příkladem je spojení dvou vstupů logického obvodu. Mnohem složitější chyba vznikne když se spojí velký počet uzlů dohromady, nebo když zkrat představuje zpětnou vazbu. V druhém případě mohou vzniknout velmi složité charakteristiky obvodu.

Vznik chybné zpětné vazby může změnit kombinační obvod v sekvenční uspořádání, a může také vyústit v nestabilitu nebo oscilace. Charakteristiky výsledného obvodu mohou být velice složité.



Obr. 2: Jeden zkrat způsobený chybou: (a) modul bez chyby, (b) chybný zkrat, (c) ekvivalentní obvody



Obr. 3: Chybné zkraty, způsobující vedení zpětné vazby: (a) obvod bez chyby, (b) možné zkratové chyby

3.2.1.3 Model neurčité chyby

Model neurčité chyby se používá k vyjádření skupiny chyb přidružených k branám CMOS, které nemohou být modelován pomocí modelu trvání jedné chyby.

Chyba nastane, když oba z vstupních tranzistorů brány CMOS jsou zavřeny jako důsledek vnitřního odpojeného obvodu, nebo zkratu. To způsobuje, že výstup není ani na úrovni 1 ani na úrovni 0, vzniklý jev je podobný stavu vysoké impedance třístavové brány TTL. Avšak oproti TTL, v tomto případě se kapacitní vliv na bráně CMOS snaží zachovat výstupní úroveň na předchozí hodnotě. V závislosti na povaze chyby může výstupní hodnoty brány ovlivňovat průběh na vstupu. Tím dostává obvod složité sekvenční charakteristiky. Doba po kterou zůstane zařízení v předchozím stavu je podmíněna kapacitancí bran připojených na výstup a ztrátami proudu.

3.2.2 Použití chybových modelů

Chybové modely jsou široce používány při návrhu testovacích schémat pro digitální systémy. Složité obvody často mívají mnoho vstupů a výstupů a velmi mnoho vnitřních součástek. Každá součástka a každé spojení je místem potencionální chyby, a výrobce je horlivý vědět, že obvod splňuje své požadavky. V jednoduchém kombinačním obvodu je možné kompletně otestovat obvod použitím všech možných kombinací na vstupu a porovnat výsledy na výstupu s předlohou. Nicméně, protože množství možných kombinací vzrůstá exponencionálně s množstvím vstupů, i testovací doba velice rychle vzrůstá se složitostí obvodu. Obvod s 25 vstupy může být otestován v několika sekundách, obvod s 50 vstupy může vyžadovat desetiletí k otestování. Tento problém je mnohem více neřešitelný v sekvenčních obvodech, kde by byly kontroly třeba na všech možných stavech.

Chybové modely poskytují metodu vypořádání se s tímto problémem charakterizováním chybových podmínek které mohou existovat uvnitř systému. Obvod s N uzly má $2N$ potenciálních chyb. Identifikací předlohy vstupů, která detekuje každou z těchto chybových okolností, je možné prověřit obvod bez vyčerpávajícího testování jeho funkčnosti. Toto je uskutečnitelné dokonce i s obvody majícími desítky tisíc bran. Kombinace použitých vstupů je označována termínem **testovací vektory**, a určení vhodné sady takových vektorů je vysoce důležité v návrhu systému.

Jeden z předpokladů použitých ve většině testovacích schémat založených na použití chybových modelů je že se vší pravděpodobností je pouze jedna chyba přítomna v obvodu v jednom časovém okamžiku. Důvod pro tento předpoklad je praktický, podle něj obvod s N uzly může mít $2N$ setrvávajících chyb, ale mohl by mít $3^N - 1$ mnohonásobných chyb. Testování pro zjištění všech možných okolností vzniku chyby může být nepraktické pro všechny obvody kromě nejjednodušších obvodů. Naneštěstí, mnohonásobné chyby mohou nastat, a nastávají, protože fyzické defekty mohou zasáhnout více než jen jeden uzel. Nicméně testování je omezeno praktičností a předpoklad jedné izolované chyby je celkem běžný.

Testování pro přítomnost zkratu způsobeného chybou nebo neurčité chyby je ještě více složité, jednak protože povaha chyb je mnohem složitější, takže i navrhnout test, který je odhalí je mnohem složitější, a za druhé kvůli potenciálně velkému množství chyb. Jak již víme, obvod s N uzly má $2N$ možných setrvávajících chyb. Jestliže je N^2 potenciálních zkratů mezi dvěma uzly, tak mezi třemi uzly je to N^3. Tyto velké hodnoty, v kombinaci se složitostí pravděpodobného chování obvodu, dělají kompletní testování ve většině případů nepraktické. Z těchto důvodů je v testovacích schématech běžné zcela přehlížet složité chybové mechanismy, a doufat, že techniky navržené k detekci jednotlivých chyb také odhalí jiné chyby.

Po více než čtvrt století byl model trvání jedné chyby používán s úspěchem, zejména když byl použit na obvody obsahující relativně jednoduché součásti jako TTL. Naneštěstí, s nárůstem používání stupňů integrace LSI a VLSI součástí převážně používající MOS technologie, začalo být zřejmým že tento model není vždy adekvátním (Wadsack, 1978, Benerjee and Abraham, 1984)[2]. Jeden z problémů modelu trvání jedné chyby je jeho spoléhání se na reprezentaci obvodu pomocí bran. Se složitými integrovanými obvody není často testovacímu inženýrovi dostupná detailní znalost vnitřní logiky. Také, jak se složitost zvyšuje, rozdíly mezi fyzickým obvodem a jeho logickým ekvivalentem se stávají významnějšími. Navzdory těmto problémům zůstává tato technologie jednou z nejpoužívanějších metod pro testování LSI a VLSI obvodů.

V bezpečnostně kritických obvodech jsme znepokojeni ne jenom přítomností nebo absencí chyby, ale i jejich dopadem. Chybové modely mohou být použity v souvislosti s **FMEA (failure modes and effects analysys)** k určení, které chyby mají za následek nebezpečné podmínky. Toho může být použito k ovlivnění návrhu tak aby se minimalizoval počet potenciálně nebezpečných stavů.

Kromě použití v testování jsou chybové modely taky důležité při výrobě chybově tolerantních systémů. Za účelem návrhu systému schopného tolerance chyb je nebytné mít představu, jak může být chyba přítomna. Chybové modely tento přehled poskytují a také umožňují návrháři odhadnout chování návrhu určením zlomku možných chyb v systému které mohou být tolerovány.

3.2.3 Chyby v návrhu a specifikaci hardwaru

Techniky modelování chyb popsané výše mohou být použity k vyjádření fyzických defektů v obvodu, ale nezabývají se příčinou těchto defektů. Součástky pracující ve vhodném prostředí se projevují náhodným selháním, a statistické techniky mohou být často použity k předpovězení pravděpodobnosti těchto selhání. Chybové modely tak mohou být použity k vytvoření strategie pro detekci nebo toleranci těchto náhodných selhání. Když je součástka pod zvýšenou zátěží, způsobenou chybou v návrhu nebo v obvodu, může selhat mnohem rychleji než předpověděla statistická analýza. Jestliže je například rezistor, schopný odvádět výkon jeden watt, přinucen odvádět dva watty, je pravděpodobné, že obvod bude po nějakou dobu pracovat správně, avšak spolehlivost tohoto rezistoru bude pravděpodobně nižší než by se dalo očekávat od součástky pracující za předepsaných podmínek. Jestliže by z důvodů této návrhové chyby součástka selhala, účinek by byl nerozlišitelný od náhodného selhání součásti. Proto jsou techniky modelování chyb popsané výše stejně tak použitelné na náhodná selhání součástky i na selhání z důvodů chybného návrhu.

Ačkoliv některá selhání vzniklá chybou v návrhu mohou být modelovány tímto způsobem, není tomu tak vždy. Některé chyby v návrhu nezpůsobují jen zvýšení zátěže na součástce, ale mohou způsobit nesprávnou činnost systému. Například chyba v logickém obvodu může způsobit, že obvod bude při dané kombinaci vstupních hodnot dávat nesprávné výstupní hodnoty. V některých případech může tato chyba mít podobné charakteristiky jako chyba způsobená selháním součástky, takže některé kombinace na vstupech mohou způsobit správné hodnoty na výstupu zatímco jiné kombinace způsobují problémy. Nicméně jiná selhání součástí jako chyby v návrhu nemohou být jednoduše modelována, a je neproveditelné vytvořit sérii testů, která by detekovala všechny možné chyby v návrhu.

Chyby ve specifikaci systému se mohou projevit jako hardwarové nebo softwarové chyby, a mohou způsobit mnohem více znepokojující problémy pro technika. Většina z technik, použitých k dosažení bezpečnosti, je zaměřena na zajištění, že systém splní svoji specifikaci. Jestliže je tato specifikace nesprávná, potom ani bezpočet technik na detekci a toleranci chyb nevyřeší tento problém. Chyby ve specifikaci zůstávají jedním z nejnepřekonatelnějších

problémů v oblasti tolerance chyb. Všeobecně musí být tyto problémy vyřešeny pomocí jedné nebo více technik na předcházení chyb, například použitím **formálních specifikačních jazyků**.

3.3 Softwarové chyby

Je všeobecně uznávaným faktem, že všechny i ty nejjednodušší programy nevyhnutelně obsahují mouchy – hovorový výraz pro softwarové chyby. Takovéto chyby se mohou vyskytovat ve skoro neomezeném počtu forem. Jako příklad můžeme uvést:

- chyby ve specifikaci softwaru
- chyby v kódování
- logické chyby ve výpočtech
- přetečení nebo podtečení zásobníku
- použití neinicializovaných proměnných

Software neselhává náhodně a neznehodnocuje se s věkem. Proto jsou všechny selhání systematická a související s jeho návrhem. Tak jako u hardwarových chyb, modelování softwarových chyb je složité kvůli jejich rozmanitým formám. Toto, společně se složitostí většiny funkčního softwaru, dělá kompletní testování ve většině případů nemožným.
V některých případech může návrhář omezit složitost běžného postupu, čímž učiní testování proveditelným. To je ale možné pouze ve výjimečných případech.

Vědomi si určité existence softwarových chyb, nemožnosti jejich lokalizace a tudíž i jejich odstranění, musíme zvážit kroky k tolerování jejich přítomnosti. Nicméně tolerance softwarových chyb je jeden z nejžádanějších požadavků v bezpečnostně kritických systémech.

4 Pokrytí chyb

Ukázali jsme si, že bezpečnosti může dosaženo kombinací odvracení chyb, odstranění chyb, detekce chyb a tolerance chyb. V každém případě úspěch těchto technik může být měřen pomocí dosaženého **pokrytí chyb**, což bývá část možných chyb které mohou být odvráceny, odstraněny, detekovány nebo tolerovány. V praxi je velice složité vytvořit nějaký číselný odhad úspěchu odvracení chyb, ale v jiných oblastech mohou chybové modely být použity k posouzení a porovnání alternativních návrhů.

Pokrytí odstranění chyb je míra úspěchu v hledání chyb během fáze testování při vývoji systému. Chybové modely jsou použity k výběru sady vektorů pro detekci maximálního počtu chyb. Výsledkem testování by mělo být dosažení 100% pokrytí chyb vytvořením testovacích vektorů k rozptýlení všech možných defektů. Avšak chybové modely samy osobě mají nedostatky, takže nezahrnou všechny možné chyby. Většina modelů je omezena na jednu chybu a neuvažují přechodné nebo střídavé chyby. Proto je pokrytí odstranění chyb vždy nižší než úplné, což zvyšuje potřebu jiných technik ke zmírnění jejich dopadu.

Mnoho architektur pro toleranci chyb spoléhá na detekci chyb během provozu. Úspěch se kterým může systém provést tento úkol je nazýván **pokrytí detekce chyb**. Pokrytí může být odhadnuto během fáze návrhu použitím chybových modelů, ale tento odhad je omezen limity při modelování a faktory v praxi. Schopnost systému tolerovat chyby je popsána jeho **pokrytím tolerance chyb**.

Použití testování ke zjištění aktuálních hodnot, dosažených v různých formách pokrytí chyb, je nepraktické v reálných systémech, protože může vyžadovat reprodukci všech možných chyb a kombinací chyb. Odhady založené na chybových modelech dávají obrázek vlivu, který mají chyby na systém. Na ty by nemělo být nahlíženo jako na číselné odhady

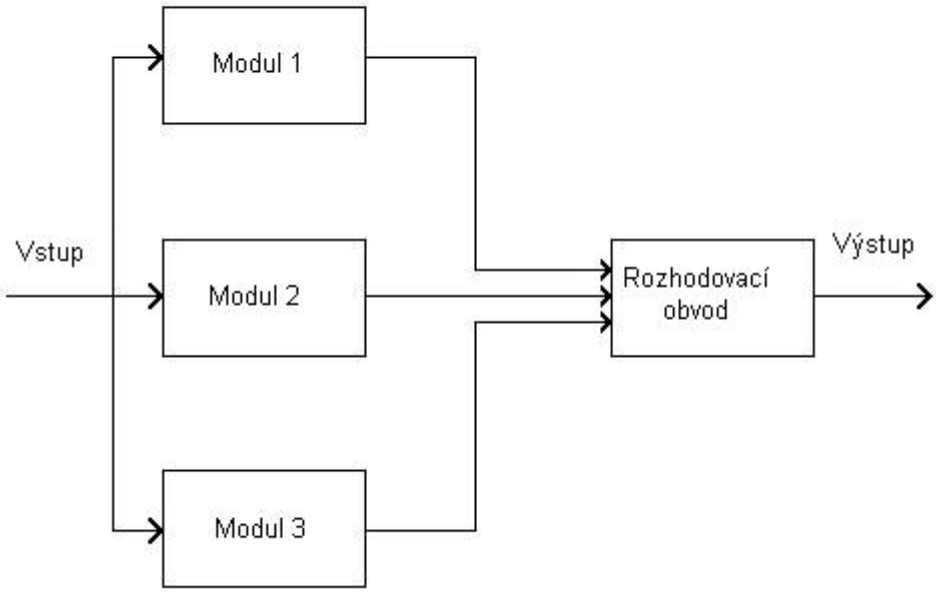
pravděpodobné spolehlivosti systému. Přínos analýzy pokrytí chyb je to, že dovoluje alternativním návrhům být srovnávány, a nebo následkům změn být prozkoumávány.

4.1 Redundance – nadbytečnost

Všech druhů chybové tolerance je dosaženo pomocí nějaké formy **redundance**, což znamená použití nějakých přídavných prvků v systému, které by nebyly třeba v systému který je naprosto bez chyb.

Většina z prvních systémů s tolerancí chyb používali zdvojené hardwarové moduly způsobem, že selhání jednoho z modulů by normálně nevyústilo selháním systému. Příkladem takového systému je jednoduchý systém tří modulů - **triple modular redundancy (TMR)**.

Systém TMR používá tři identické moduly, každý přijímá stejný vstupní signál. Když všechny moduly pracují správně, tak produkují stejný výstupní signál, a jakékoliv rozdíly mezi jejich výstupy znamenají selhání modulu. Je použit rozhodovací prvek k odstranění vlivu chybného modulu porovnáním signálů a použitím většinové hodnoty v případě nesouhlasu. Systém tímto může tolerovat selhání jednoho z modulů bez ovlivnění výstupu systému.



Obr. 4: Jednoduchý systém tří modulů

4.1.1 Druhy redundance

Systém TMR popsaný výše je jenom jedním z mnoha druhů redundance, a je vhodné rozlišit několik základných druhů.

Hardwarová redundance

Použití více hardwarových součástí než by bylo třeba použít v systému s absencí chyb, se zaměřením na detekci nebo toleranci chyb. Systém tří modulů popsaný výše je příkladem tohoto druhu redundance, a další druhy jsou popsány v kapitole o toleranci hardwarových chyb.

Softwarová redundance

Použití přídavného softwaru, který by nebyl třeba použít v systému s absencí chyb, se zaměřením na detekci nebo toleranci chyb. Metody softwarové redundance jsou popsány v kapitole o toleranci softwarových chyb.

Informační redundance

Použití přídavné informace kromě té, která je vyžadována v implementaci dané funkce, se zaměřením na detekci a toleranci chyb. Příkladem může být použití paritních bitů, detekčních nebo korekčních kódů a kontrolních součtů. Informační redundance může být implementována použitím hardwarových nebo softwarových technik, a je široce použita v komunikačních a VLSI zařízeních, jako jsou paměti a procesory.

Časová redundance

Použití většího množství času, než je vyžadováno danou funkcí, se zaměřením na detekci a toleranci chyb. Časová redundance může zahrnovat opakování výpočtů a následné porovnání výsledků. To může být použito k detekci přechodných chyb, a pokud jsou provedeny více než dva výpočty, je možno jeden chybný výpočet ignorovat, a tím provést toleranci chyb. Časová redundance může být provedena použitím hardwarových nebo softwarových technik.

Praktický systém s tolerancí chyb používá rozumnou směsici technik k poskytnutí ochrany proti škále možných chyb.

4.2 Diverzita návrhů

Systém TRM popsaný dříve poskytuje jistou ochranu proti náhodnému selhání hardwarové součásti ale žádnou ochranu proti chybám v návrhu, protože stejné moduly se pravděpodobně projevují stejnými problémy. Selhání jako důsledek podobných chyb v různých nadbytečných modulech jsou nazývána **selhání společného módu**. Provádění ochrany proti těmto chybám je mnohem obtížnější než proti chybám náhodného selhání komponent. Této ochrany nemůže být dosaženo jednoduše zdvojením stejných systémů. Základní problém návrhových chyb je nemožnost předpovědět povahu nebo místo chyby. Většina pokusů vypořádat se s těmito problémy počítá s redundancí kombinovanou s nějakou formou **diverzity**.

V odlišných systémech je provoz vykonáván různými způsoby v naději, že stejná chyba nebude přítomna v různých provedeních. Příkladem tohoto způsobu může být TMR systém popsaný nahoře, kde by byli jednotlivé moduly navrhnuty a vyrobeny rozdílnými týmy ve snaze o snížení možnosti vzniku stejných problémů na jednotlivých modulech. Tento přístup poskytuje zvýšenou ochranu proti chybám společného módu výměnou za velké zvýšení nákladu návrhu.

Diverzita návrhů může být použita ve všech formách redundance. V hardwarové části může zahrnovat použití modulů různého designu, kdežto v softwarové části to může vyžadovat různé programy k implementaci stejné funkce. Je často výhodou zvolit různé technologie k realizaci jednotlivých kanálů redundantního systému. Například jeden kanál může používat systém založený na mikropočítači, kdežto paralelní systém může být založen na koncepci pevných obvodů. Kde je implementace bez možnosti naprogramování nepraktická, může být vysokého stupně diverzity dosaženo použitím mikropočítače v jednom

kanálu a programovacího logického kontroléru (PLC) v druhém. Použití těchto dvou odlišných koncepcí vede k velmi rozdílnému hardwaru a softwaru uvnitř dvou implementací.

V ideálním světě by systém obsahující redundanci i diverzitu mohl poskytnout dobrou ochranu proti náhodnému selhání součástky a některým druhům chyb v návrhu. A kdyby ne, tak aspoň poskytne ochranu proti chybám ve specifikaci, když i rozdílné systémy se budou všeobecně snažit o implementaci stejné specifikace.

I když snižuje problémy přidružené k chybám ve specifikaci, použití diverzity nemůže úplně odstranit možnost selhání společného módu. Výzkum ukázal že použití různých týmů při návrhu modulu neodstraňuje pravděpodobnost stejných chyb v implementaci. Jeden konkrétní výzkumný projekt zahrnoval práci 27 inženýrů, kteří nezávisle vytvořili program podle stejné specifikace. Výsledky ukázaly, že ačkoliv byly programy extrémně spolehlivé, obsahovali mnoho běžných chyb (Knight and Levenson, 1986) [2]. Navzdory tomuto závěru je všeobecně uznávaným faktem že diverzita návrhů má velký význam ve snižování selhání společného módu, ačkoliv nemůže garantovat úplné odstranění těchto problémů.

4.3 Techniky detekce chyby

Mnoho technik tolerance chyb se spoléhá na detekci chyb. V praxi jsou chyby detekovány jako selhání, která způsobují, a proto by jsme možná měli mluvit o detekci selhání než o detekci chyb. V některých aplikacích je vhodné uvažovat spíše selhání než chyb. To je částečně pravda když uvažujeme přechodné chyby které způsobí selhání a pak zmizí. Jakýkoliv pokus o lokalizaci chyby by byl bezvýsledný, ale je důležité vypořádat se s následky chyby v působení výsledného selhání. V oblastech jako jsou komunikace, selhání jsou často představována přechodným šumem. Zde je běžné být znepokojen detekcí chyb a proto se používají techniky jako detekční a korekční kódy ze zmírnění vlivu chyb. V jiných oblastech je většina chyb permanentních, a tak je důležité lokalizovat zdroj chyb čímž mohou být učiněny vhodné kroky ke zmírnění vlivu následků chyb. Nás zajímají především chyby uvnitř systému, a proto se budeme zabývat především termínem detekce chyb.

Chyby se mohou vyskytovat ať už v hardwarové tak i v softwarové části. Stejně tak můžeme použít i hardwarové i softwarové techniky k lokalizaci chyb obou typů. Jakmile je chyba detekována, tato informace může být použita se správným druhem hardwarové nebo softwarové tolerance chyb. Je používáno velké množství schémat pro detekci chyb. Uvedeme si několik příkladů:

<u>Kontrola funkčnosti</u>

Vědomi si toho, že softwarové techniky mohou být použity k detekci hardwarových problémů, začneme se dívat na kontrolu funkčnosti. To zahrnuje použití rutinních postupů kontroly ke zjištění korektní funkčnosti hardwaru. Běžnými příklady jsou kontroly paměti, procesoru a komunikačních obvodů.

Paměť s náhodným přístupem – Random-access memory (RAM) vyžaduje při testování zápis a následné čtení částí paměti pro ověření její funkce. V malých systémech může být proveditelné testování všech částí, ale ve většině případů je testován jen zlomek paměti. Některé postupy musejí jít do návrhů algoritmů použitých k testování paměti, protože jednoduché obdržení výsledku *zapsáno* není vždy potvrzením, že zařízení pracuje správně. Například když zmíněná lokace v paměti má trvalou chybu na jednom z bitů, ale použitý testovací model odpovídá chybnému bitu. Taková chyba může být detekována testováním lokace v paměti dvakrát použitím vzoru a jednoho jeho doplňku, i když takové testování není jednoduché. V některých případech testování neexistující lokace v paměti použitím této techniky nedetekuje absenci zařízení. Kapacitance sběrnice je často dostatečná k uchování

zapsaného datového slova, takže je zase při dalším přístupu přečtena. Takové problémy mohou být vyřešeny prokládáním zápisů a čtení rozdílnými lokacemi. Během let bylo vyvinuto několik velice účinných algoritmů pro testování velkých paměťových úseků v krátkém čase, s rozumným pokrytím chyb.Tyto testy jsou často použity přidání kontroly parity nebo jiných druhů redundance informace. Paměť pouze ke čtení – Read-only memory (ROM) může být testována periodickým výpočtem kontrolního součtu a porovnáním těchto údajů s hodnotami uloženými v ROM.

Procesor nebo procesory v systému mohou být zkontrolovány vykonáním sekvence výpočtů a porovnáním výsledků se známými hodnotami. Data a očekávané výsledky těchto výpočtů jsou uloženy v ROM. Pokud kontrolní postup obsahuje širokou škálu operací na zpracování dat, tak tento test ověřuje provoz velké části systému, včetně procesoru, částí paměti a systému sběrnic.

V multiprocesorových systémech mohou být kontroly provedeny periodicky k ujištění se, že každý procesor může komunikovat se sousedními procesory. Tímto způsobem lze ověřit komunikační cesty a také umožňuje jednomu procesoru detekovat selhání druhého.

Kontrola konzistence

Kontrola konzistence používá některé znalosti povahy informace v systému k otestování její platnosti. Příkladem tohoto druhu testování je **kontrola rozsahu**. Porovnávají se vypočtené nebo uložené hodnoty pro proměnnou s předdefinovanými hodnotami pro povolený rozsah.

Porovnání signálu

V systémech s redundancí je možné kontrolovat signál na shodných bodech různých modulů, a tím potvrdit jejich funkčnost. Tento proces je jednodušší když jsou moduly identické, než když jsou různého návrhu.

Kontrola párů

Kontrola párů je speciální případ použití porovnání signálů. Zde jsou identické moduly navrhnuty k porovnání několika signálů jako pokus o detekci jakýchkoliv rozporů. Pokud moduly produkují identické signály tak se předpokládá, že oba jsou bez chyb.

Redundance informace

Různé formy redundance informace mohou být použity k detekci chyb. A to například kontrolou parity, kontrolními součty, cyklickými redundantní kódy a chyby opravujícími kódy. Každá z těchto technik používá přidanou redundantní informaci, která je použita ke zkontrolování platnosti dat.

Monitorování instrukcí

Normální operace procesoru zahrnují opakovaný přenos a vykonání instrukcí. Chyba během přenosu instrukce může poškodit operační kód nebo vstupní data. V obou případech je pravděpodobné, že nebude správná operace provedena.

Poškození vstupních dat všeobecně způsobí provedení jedné z instrukcí, i když té nesprávné. Modifikace operačního kódu někdy vyprodukuje nesmyslnou hodnotu, kterou procesor může detekovat jako chybu. Akce, kterou procesor provede v reakci na takovou

chybu se mezi různými zařízeními značně liší. Některé procesory okamžitě vytvoří výjimku, čímž umožní aby byla chyba detekována dříve než dojde k nějakému poškození. Jiné procesory mají méně vyhovující reakce, jednoduše vyvolají další byt z paměti přičemž zřejmou chybu systému ignorují. V mnoha zařízeních je akce podniknutá za těchto okolností nedefinována. Procesor, který neprovede správnou akci v reakci na nevykonané instrukce nesmí být použit v bezpečnostně kritických systémech.

Testování smyčky

Jak bylo popsáno dříve, mnoho setrvávajících chyb vzniká díky signálovým vodičům, který jsou přerušeny nebo zkratovány na jinou část obvodu. Jedna metoda detekce chyb tohoto druhu je testování smyčky. Tím lze ověřit, že signál opouštějící jeden bod obvodu přijde do bodu určení nezměněný. Toho je dosaženo poskytnutím nezávislého vodiče pro návrat signálu do zdroje a porovnáním odcházejícího a vracejícího se signálu k zajištění ekvivalence.

Testování smyčky je široce využíváno na vstupně/výstupních obvodech, konkrétně v aplikacích jako je sériová komunikace. Může být také použito samotné procesorové části, k zajištění správných spojení mezi procesorem a přidruženými komponentami. Například sběrnice mohou být konstruovány jako smyčky, takže signál opustí procesor, jde do všech důležitých uzlů obvodu a pak se vrací do procesoru k ověření. V systémech s více deskami plošných spojů může být použita soustava tzv. “věnečku” k zjištění, zda jsou všechny desky přítomny. Efektivita této techniky vyžaduje velikou péči při návrhu desky k zajištění, že odchozí a návratové vodiče spolu nikde nesousedí, a tím zabránění možnosti zkratu.

Časový spínač - hlídací pes

Jedna z nejjednodušších metod detekce selhání procesoru je použití časového spínače. Časovač je nastaven tak, že pokud je dovoleno přerušení tak způsobí restart systému, ale zároveň chrání systém před restartem způsobeným specifickou aktivitou procesoru. Pokud procesor pracuje normálně, do časovače je periodicky nahrávána příslušná hodnota. Časovač opakovaně snižuje tuto hodnotu a, pokud je cyklus neporušen, může eventuálně dosáhnout nuly a restartovat systém. Hodnota, která je nahrávána do procesoru je však zvolena tak, aby byla obnovena procesorem dříve, než dosáhne nuly. I když by procesor selhal, došlo by k zastavení obnovování časovače, který by brzo přerušil cyklus a způsobil by restart systému.

Ačkoliv je snadné ho realizovat, má hlídací pes omezení. Ovlivněn selháním systému, procesor často pokračuje v činnosti po značnou dobu než zareaguje hlídací pes. Během této doby je provoz procesoru nepředpovídatelný a potenciálně nebezpečný. Ačkoliv hlídací pes normálně zareaguje během několika milisekund, někdy to nemusí být dostatečně rychle k zajištění bezpečnosti nebo k odvrácení značného poškození databáze systému. Je také možné, že systém selže způsobem, kdy je hlídací pes stále periodicky restartován a tím zabraňuje svému vlastnímu zásahu.

Monitorování sběrnice

Více důmyslnou metodou kontroly provozu procesoru je monitorování sběrnice pro zjištění rozsahu adres získaných programem. Každá adresa ve sběrnici je porovnána s povoleným rozsahem adres pro daný program a jakákoliv hodnota mimo rozsah způsobí hlášení chyby procesoru.

Monitorování napájení

Mnoho moderních elektronických součástek pracuje s celkem úzkým rozsahem napájecího napětí. Naštěstí je absolutní maximální napětí povolené pro takové součástky značně větší než jejich nominální provozní napětí. Proto kvalitně navržené napájení spolu s ochranou proti přepětí může normálně ochránit součástky před poškozením nadměrným napájecím napětím.

Mnohem větší problémy mohou vzniknout když napájecí napětí klesne pod hodnotu potřebnou pro normální provoz systému. Ačkoliv může napájecí soustava ochránit systém před nadměrným napájecím napětím, nemůže zabránit poklesu napětí pod přijatelnou úroveň. To je nevyhnutelné, když je systém poprvé zapnut a vypnut. K tomu může také dojít, když napájecí napětí poklesne během provozu. Mnoho procesorů se za těchto okolností chová nepředvídatelně, i když paměť a adresace v obvodu může fungovat správně. To může mít za následek velké množství poškozených dat.

Nepříznivé efekty kolísání napájení mohou být sníženy použitím monitoru napájení. Ten detekuje pokles napětí před tím, něž dosáhne nebezpečné úrovně, a poskytuje procesoru potřebný čas k podniknutí nouzových opatření před tím než napájení úplně zmizí. Data jsou ochráněna zabráněním provozu procesoru během doby nízkého napětí – třeba udržením ho ve stavu restartu. Tyto podmínky jsou udržovány dokud se napájení nevrátí na přijatelnou úroveň.

Některé systémy musejí pracovat nepřetržitě a nemohou být narušeny selháním napájení. V takových případech musí být použit **nepřerušitelný zdroj napětí**. Takový zdroj používá baterie o velké kapacitě, které jsou nepřetržitě nabíjeny během normálního provozu, čímž poskytují energii v případě selhání napětí

4.4 Tolerance hardwarových chyb

Nejpoužívanější metoda dosažení chybové tolerance zahrnuje použití nadbytečného hardwaru. Jak se snižují ceny a rozměry elektronických součástek, je použití reduplikovaného hardwaru stále více výhodnější a se zvyšující měrou se využívá jako metoda zlepšení spolehlivosti systému.

Hardwarová redundance se používá ve třech základních formách, a to statická, dynamická nebo hybridní. **Statické** systémy spíše než detekci chyb využívají maskování chyb k dosažení chybové tolerance. Jsou navrženy k toleranci chyb bez nutnosti zásahu systému nebo operátora. **Dynamická** redundance se spoléhá na detekci chyby a na systém který podnikne příslušnou akci k odstranění jejích následků. To znamená rekonfiguraci systému k odstranění vlivu chybné součástky. **Hybridní** techniky používají kombinaci těchto metod. Hybridní přístup používá maskování chyb jako prevenci a detekci chyb s rekonfigurací k odstranění chybné jednotky ze systému.

4.4.1 Statická redundance

Statická redundance se spoléhá na použití nějakého druhu rozhodovacího mechanismu pro porovnání výstupů všech modulů a zamaskování chyby v těchto modulech. Nejjednodušší verze tohoto systému vyžaduje tři moduly a je nazývána systémem tří redundantních modulů. Zvýšení chybové tolerance může být dosaženo použitím dalších redundantních modulů. Takové systémy jsou všeobecně nazývány systémem N redundantních modulů.

Trojmodulová redundance

Jednoduchá trojmodulová redundance TMR-(Triple modular redundancy) je zobrazena na obr.4. V systémech jako je tento přijímají tři moduly stejný vstupní signál a měli by produkovat stejný výstupní signál. Rozhodovací obvod porovnává výstupy z těchto tří modulů. Pokud všechny tři moduly souhlasí, je tento signál přímo předán dál rozhodovacím obvodem. Když, jako důsledek jedné chyby, je výstup jedné jednotky odlišný od ostatních, pak rozhodovací obvod vyprodukuje na výstupu signál odpovídající většinovému signálu. Toto většinové rozhodování tím maskuje selhání jednoho modulu.

Zde je redundance použita k zabránění selhání celého systému selháním jedné součástky, takzvanému **selhání jednoho bodu**. Ztrojení modulů sice chrání před selháním v modulu, ale neřeší selhání jiných míst v obvodu. Jedním z takových míst je zdroj vstupního signálu. Když je tímto zdrojem jeden senzor, pak by selhání tohoto senzoru ovlivnilo vstupy do všech modulů a tím by se potlačil efekt použití redundance. V kritických systémech je běžné senzory zdvojit, nebo i ztrojit jako prevence před možným selháním jednoho bodu. V takových případech není neobvyklé požít různé senzory ke snížení pravděpodobnosti systematické chyby v senzorech způsobující běžná selhání.

Naneštěstí použití více senzorů může někdy způsobit jiné problémy. Obzvláště v případě analogových zdrojů signálů, nebo senzorů s rozdílnou rychlostí odezvy. Když je analogový signál přiveden na vstup digitálního systému, tak musí být digitalizován použitím analogově-digitálního převodníku. K zamezení možného zdroje selhání v jednom bodě je vhodné zdvojit ne jenom analogové senzory, ale také k nim přidružené analogově-digitální převodníky. Bohužel kvantizační zkreslení způsobí, že všeobecně tyto převodníky ne vždy produkují identický výstupní signál, což značně komplikuje porovnávání výstupů z modulu. Senzory s různou dobou odezvy způsobují podobné problémy. Protože zdvojené senzory musejí být fyzicky odděleny, tak jsou všeobecně aktivovány v nepatrně jiném čase. Kvůli vysoké rychlosti digitálních systémů mohou rozdíly mezi výstupy senzorů způsobit nesouhlasy na výstupech modulů. Tyto rozdíly v trvání přenosu je nutno také ošetřit.

Dalším místem náchylným k selhání jednoho bodu je rozhodovací prvek. Jedním z přístupu k tomuto problému je navrhnout spolehlivý rozhodovací obvod. Funkce rozhodovacího prvku nejsou složité a proto by mělo být možné vytvořit jednoduchou, spolehlivou jednotku, která bude splňovat celkové provozní požadavky systému. Další alternativou je znásobení rozhodovacích obvodů.

Ztrojení rozhodovacího obvodu odstraní zdroj selhání jednoho bodu a vytvoří tři nezávislé výstupy, které budou všechny správné za předpokladu, že nedojde k selhání více než jen jednoho modulu. Několik takových stupňů může sestaveno do kaskády, kde jsou výstupy z jedné trojice zapojeny na vstup do druhé trojice. Selhání jednoho rozhodovacího prvku způsobí selhání jednoho výstupu, a to pak je odstraněno dalším TMR stupněm.

Je jasné, že TMR nemůže poskytnout ochranu před současným selháním dvou ze tří modulů. Použití této techniky se opírá o předpoklad, že selhání dvou modulů je mnohem méně pravděpodobné než selhání jednoho modulu. Náhodné selhání hardwarové součástky a systematická chyba, třeba v návrhu hardwaru nebo softwaru, může postihnout všechny stejné moduly ve stejný moment. Hardwarová redundance používající identické moduly poskytuje pouze malou ochranu proti těmto systematickým chybám.

Statická hardwarová redundance poskytuje chybovou toleranci maskováním vlivu selhání modulu. Chyby jsou proto tolerovány bez nutnosti podniknutí nějaké akce systémem. Nicméně, jakmile modul selže, tak schopnost systému tolerovat další selhání je snížena, v TMR systémech může být zrušena. Proto je nezbytné zaznamenat přítomnost chybného modulu aby mohl být opraven nebo vyměněn. To může konkrétně být důležité v případě občasné nebo přechodné chyby, která nemusí být zaznamenána během pravidelné údržby.

Systém, který zaznamenává každý nesouhlas mezi moduly, poskytne velice užitečná data k detekci nespolehlivých modulů nebo operací v systému.

N – modulová redundance

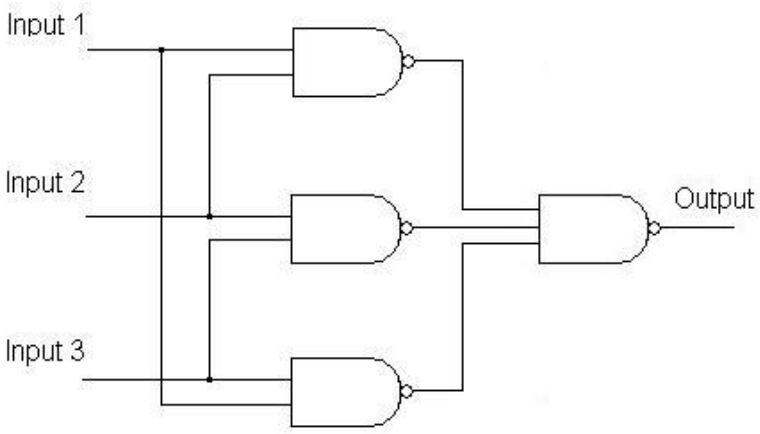
TMR systém může být upraven rozšířením na jakýkoliv počet modulů. Tento systém je všeobecně nazýván ***N* - modulová redundance** – NMR (*N*-modular redundancy), a v mnoha případech je použit sudý počet modulů k zajištění většinového schématu rozhodování.

Jak se zvyšuje počet modulů, tak i schopnost systému čelit selhání modulu se zvyšuje. Všeobecně selhání modulu nezpůsobí selhání systému za podmínky že většina modulů pracuje správně. Proto, jestliže systém se třemi moduly může tolerovat jenom selhání jednoho modulu, pak systém s pěti moduly může tolerovat dvě selhání a systém se sedmi bude tolerovat selhání tří modulů. Systém bude vždy tolerovat selhání $\frac{N-1}{2}$ modulů bez vzniku systémové chyby. Výhody použití velkého množství redundantních modulů jsou jasné. Nicméně nevýhody jsou také evidentní v nárocích na vysokou cenu, rozměry, váhu a spotřebu elektřiny. V praxi je počet použitých modulů málokdy vyšší než čtyři. Stejně jako v systému TMR můžeme použít jeden rozhodovací obvod, ale výhody použití více rozhodovacích obvodů jsou také jasné.

Techniky rozhodování

Operace rozhodnutí může být provedena jak v hardwarové části tak i v softwarové části, a může proběhnout v několika úsecích v systému.

Hardwarový rozhodovací prvek může být velice jednoduchý. Obr. 5 znázorňuje sestavu vhodnou pro TMR systém. V něm se porovnávají tři vstupy a vzniká výstup logická 1 jestliže dva nebo více vstupů jsou 1, nebo logická 0 jestliže dva nebo více vstupů je 0. Znázorněné pravdivostní tabulka této sestavy demonstruje charakteristiku obvodu. Díky malé složitosti může být obvod mnohem spolehlivější než ten se zdvojenými moduly. V mnoha případech moduly produkují mnoho bitů vyžadujících několik nezávislých srovnání. Toho může být dosaženo použitím tohoto jednoduchého obvodu na každý výstupní bit. Může být zřejmé, že ačkoliv obvod samotný je jednoduchý, pro obvody produkující velké množství výstupů představuje rozhodovací obvod významnou část hardwaru. Je nutné také zmínit, že tento jednoduchý obvod nemá žádný mechanismus pro indikaci nebo záznam odchylek pro použití během údržby nebo pro zhodnocení spolehlivosti. Zahrnutí takových prvků do



Pravdivostní tabulka

Vstup			Výstup
1	2	3	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Obr. 5: Jednoduchý tří-vstupový, jednobitový rozhodovací prvek

hardwarového rozhodovacího obvodu způsobí značné zvýšení jeho složitosti a odpovídající snížení spolehlivosti.

Rozhodování v softwarové části odstraňuje potřebu dodatečného hardwaru. Také zjednodušuje shromažďování dat vzniklých díky nějakému nesouhlasu mezi vstupy. Tyto informace jsou druhem detekce chyb a jsou neocenitelné při určování požadavků údržby. Mělo by být pamatováno, že software nevzniká zadarmo a že implementací jakýchkoliv extra prvků do softwaru vznikají další náklady při vývoji, rychlosti odezvy systému a zvýšení složitosti systému. Provoz monitorování chyb v softwaru zvyšuje jeho složitost, a tím má také dopad na spolehlivost systému.

Jednou ze základních úvah v rozhodnutí mezi hardwarovým a softwarovým rozhodovacím systémem je posouzení požadované rychlosti provozu. Hardwarové rozhodovací prvky pracují velice rychle, jsou omezeny pouze zpožděním přenosu mezi branami. Softwarové rozhodovací prvky jsou mnohem pomalejší, protože každé porovnání může znamenat mnoho operací procesoru. V mnoha případech je vhodná technologie zvolena na kompromisu mezi rychlostí provozu a hardwarovými náklady.

4.4.2 Dynamická redundance

Statický systém dosahuje chybové tolerance pomocí **maskování** chyb, a tímto způsobem brání šíření chyb v systému. Cenou za to je vysoká úroveň redundance, kdy jsou nejméně tři moduly potřebné k toleranci jedné chyby a pět modulů k ošetření dvou chyb. Dynamické systémy používají rozdílný přístup, používají detekci chyb místo maskování chyb. Jedna jednotka je normálně aktivní, a jeden nebo více dalších **záložních systémů** bývá k dispozici v případě že tato jednotka selže. Tento přístup snižuje množství potřebné redundance, protože jsou pouze dva moduly potřebné ke zvládnutí jedné chyby a tři k zajištění dvou chybných jednotek. Úspěch tohoto přístupu je značně podmíněn efektivitou procesu detekce chyb.

Jak bylo zmíněno dříve, schémata detekce chyb pracují na principu detekce chyby systému způsobené chybou součástky. Proto musí chyby vzniknout v systému před tím, než může dynamický systém rozpoznat chybu a podniknout příslušnou akci. Dynamické systémy nemaskují chyby, ale spíše se snaží zvládnout chyby a potom překonfigurovat systém k dosažení chybové tolerance. Z toho důvodu je dynamická redundance vhodná v aplikacích, které mohou tolerovat chyby v průběhu provozu.

Záložní systém

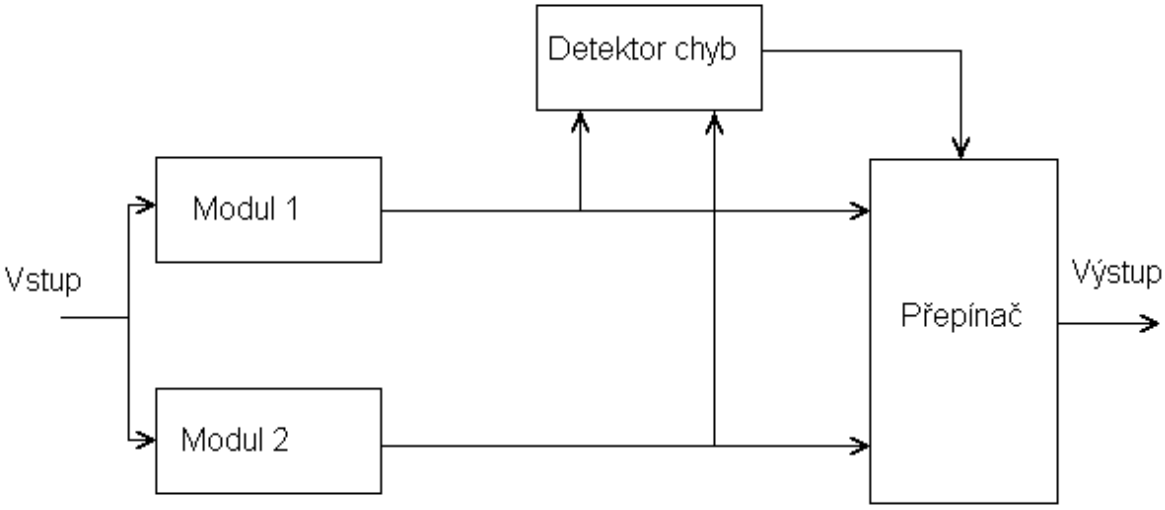
Jeden z nejpoužívanějších dynamických redundantních systémů je záložní systém, jak je zobrazeno na Obr. 6. Zde jeden modul pracuje ve spojení s nějakým typem detekce chyb. Když není detekována žádná chyba, tak jeden modul posílá svůj výstupní signál skrze přepínač který je řízen systémem detekce chyb. Pokud je detekována chyba, tak přepínač rekonfiguruje systém tím, že přepne na výstup ze záložního systému.

Rekonfigurace systému způsobená detekcí chyby efektivně odstraní chybný modul ze systému a tak odstraní její vliv. Proces rekonfigurace systému způsobí během přepínání výstupů krátkodobý výpadek systému. Výpadek může být minimalizován použitím **teplého záložního systému**, kde náhradní obvody běží souvisle paralelně s aktivní jednotkou. To umožňuje rychlé přepnutí kontroly s minimálním zpožděním. Nevýhodou tohoto systému je, že záložní systém zvyšuje nároky na napájení systému. Záložní jednotka je také vystavena stejnému provoznímu zatížení jako aktivní jednotka. Alternativním přístupem je použití **studeného záložního systému**, kde jsou záložní obvody zapojeny v momentě, kdy jsou třeba. To snižuje nároky na spotřebu elektřiny a zmenšuje opotřebení záložního modulu, ale

všeobecně způsobí větší výpadek při přepínání. Provoz nemůže být přepnut na náhradní jednotku dokud není zapnuta a nejsou provedeny inicializační procesy.

Dvou modulový systém na Obr. 6 může být rozšířen na jakékoliv množství záložních modulů. Zde opět jeden modul dává výstupní signál, a obvody detekce chyb přepínají na další moduly v případě, že dojde k selhání modulu.

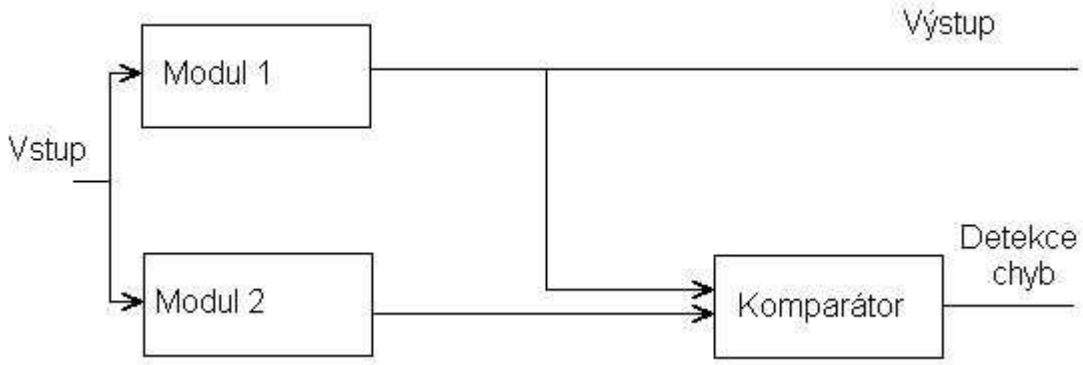
Efektivita záložního systému velice závisí na provedení použité techniky detekce chyb.



Obr. 6: Uspořádání záložního systému

Páry se automatickou kontrolou

Dalším příkladem dynamické redundance je systém párů s automatickou kontrolou. Zde je na dva identické moduly přiváděn stejný vstupní signál, a jejich výstupy jsou porovnávány. Výstup z jednoho modulu pokračuje do dalšího úseku, a výstup z komparátoru je použit jako signál detekce chyb. Uspořádání je zobrazeno na Obr. 7.

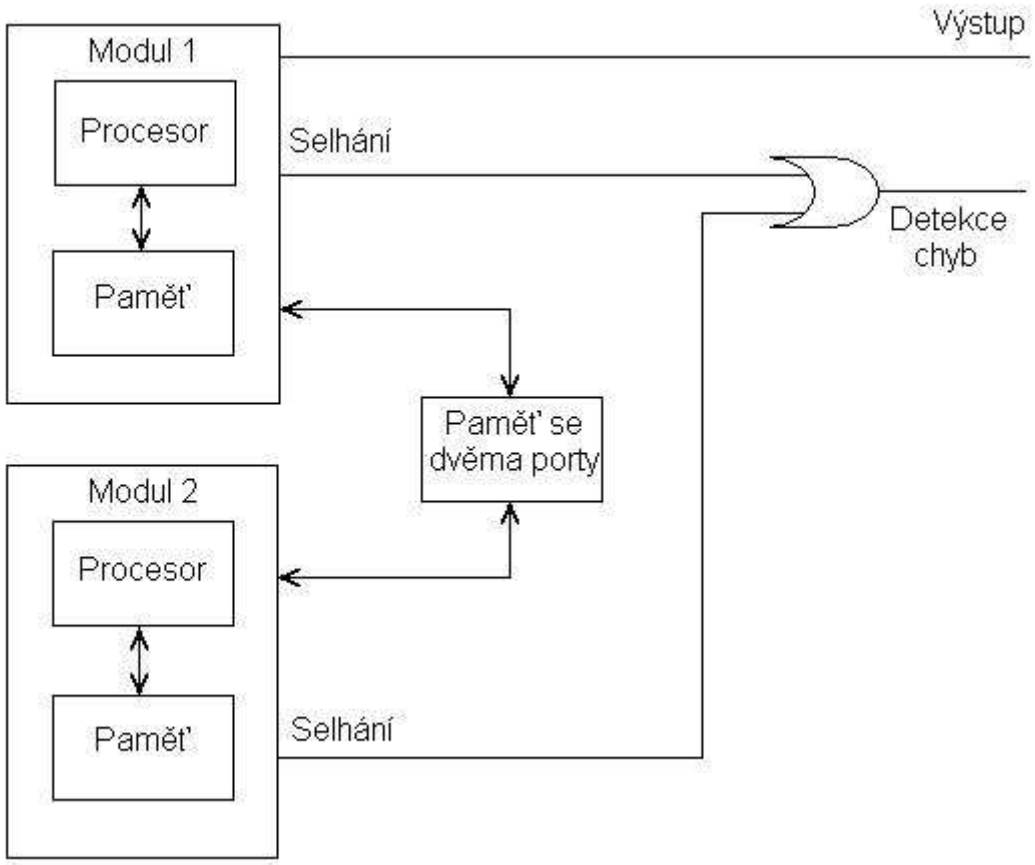


Obr. 7: Páry s automatickou kontrolou

Komparátor může být implementován v hardwaru, i v softwaru. Jednoduchý hardwarový komparátor může být vytvořen použitím několika XOR bran. Jedna brána je použita k porovnání odpovídajících si signálů ze dvou modulů. Výstup ze XOR brány je 0 když jsou signály stejné a 1 když jsou rozdílné. Výstup ze sady bran může být OR brána, která produkuje signál 1 jestliže jakékoliv porovnání selže. To nám formuje signál detekce chyb.

Jestliže každý z modulů v páru obsahuje procesor, pak může být porovnání provedeno v softwaru, což odstraňuje potřebu dodatečného hardwaru. K odstranění možnosti selhání jednoho bodu musejí oba procesory provést porovnání a proto musejí mít oba přístup k signálu a k výsledkům z druhého procesoru. Toho lze dosáhnout mnoha způsoby, například způsobem na Obr. 8. Obvod zde používá paměť se dvěma porty pro komunikaci mezi dvěma moduly. Procesory potom porovnávají jejich výsledky s výsledky z dalšího modulu a vygenerují signál „selhání“ jestliže je detekován nějaký rozdíl. Chybový signál ze dvou modulů je zkombinován v bráně OR, který produkuje signál „detekce chyby“ jestliže nějaká jednotky selže. Selhání paměti se dvěma porty bude normálně detekována oběmi jednotkami a to způsobí rozdíly mezi uloženými a vypočtenými hodnotami.

Dvojitého porovnání provedené softwarem v tomto systému může být také dosaženo v hardwaru, ale za náklady dodatečného hardwaru. Opatření mnohonásobných porovnání snižuje pravděpodobnost, že chyba v modulu způsobí nesprávný výstupní signál kvůli selhání komparátoru. Výsledkem tohoto zdvojení je produkce více signálu indikujících, že byla detekována chyba. Na Obr. 8 jsou dva signály kombinovány použitím OR brány. Selhání této brány může způsobit ztrátu funkce detekce chyb navzdory zdvojení komparátorů. Tato slabina může být překonána uspořádáním na Obr. 9. Zde výstupy ze dvou nezávislých komparátorů ovládají oddělené spínače v sérii s výstupem z aktivní jednotky. Jestliže obě porovnání neukáží žádné rozdíly, tak jsou oba spínače zavřeny a výstup z aktivní jednotky je aplikován v místě určení. Jestliže některé z porovnání ukáže nesouhlas mezi moduly, jeden ze spínačů se otevře a výstup bude odpojen. Tak musejí oba moduly souhlasit aby byl na výstupu nějaký signál. Selhání buď komparátoru nebo spínače by nemělo pustit nesprávný signál na výstup

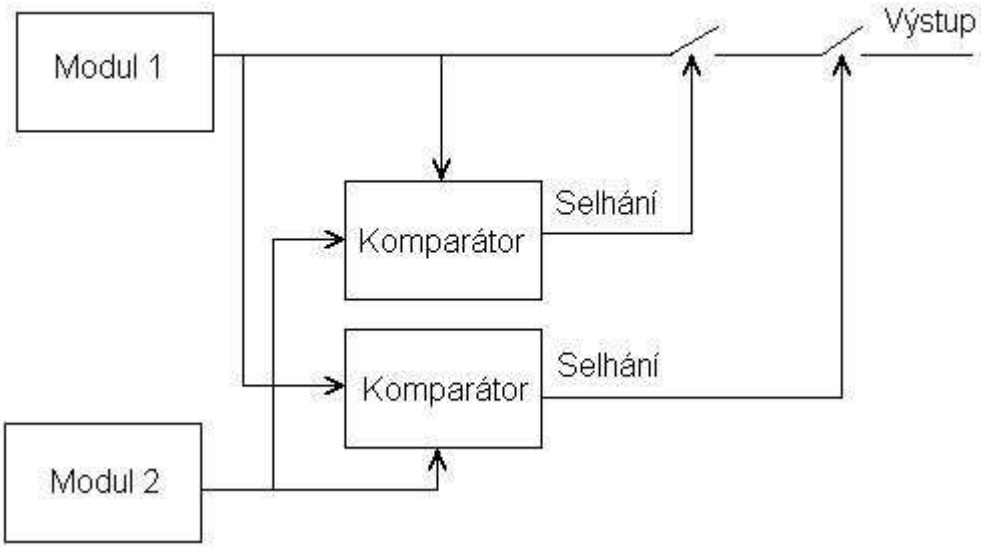


Obr. 8: Páry s automatickou kontrolou používající softwarové porovnání

ani v případě selhání modulu. Takové uspořádání není vhodné ve všech situacích, ale tam, kde odpojení výstupu způsobí, že se systém přepne do bezpečného módu.

Uspořádání párů s automatickou kontrolou neposkytuje samo o sobě toleranci chyb. Nicméně poskytuje výkonnou metodu detekce chyb, která může být použita ke zvýhodnění dynamických systému s tolerancí chyb. Například může být několik párů s automatickou detekcí použito v konfiguraci záložního systému, s porovnáním provedeným v párech a tím dávající základ pro systém detekce chyb.

Při popisování systémů statické redundance je zmíněno, že analogový vstup a signály s proměnou dobou odezvy způsobují rozhodovací proces složitý, protože rozdílné moduly nemusejí vždy odečíst stejný vstup. Takové vstupy taky způsobují problémy párům s automatickou kontrolou z těch samých důvodů. Vypořádání se s těmito problémy může často vést ke značnému zvýšení složitosti systému.



Obr. 9: Sloučení signálů detekce chyb použitím spínačů

4.4.3 Hybridní redundance

Ukázali jsme si, že statická redundance používá rozhodování jako maskování chyb, ale toho dosahuje za cenu velkého množství redundance. Dynamické systémy používají detekci chyb a nějaký druh spínání k odstranění chybné jednotky ze systému. Tento přístup vyžaduje méně redundance, ale nemaskuje chyby, čímž ho to dělá nevhodným v mnoha aplikacích kde jsou přechodné chyby nepřijatelné. Hybridní systémy kombinují tyto techniky a může poskytnout mnoho výhod těchto dvou přístupů.

Hybridní redundance používá kombinaci rozhodování, detekce chyb a přepínání modulů. Používá se mnoho technik, ačkoliv většinu z nich lze shrnout jako nějaký druh *N* - modulové redundance.

N-modulová redundance se zálohou

Tento přístup zahrnuje prvky ze statické techniky *N* - modulové redundance a dynamických metod založených na použití záložních systémů.

Toto ztvárnění poskytuje *N* aktivních modulů spojených skrze spínací část k rozhodovacímu obvodu. V případě, že nedojde k chybě v modulu, tak rozhodovací obvod bude jednoduše předávat jednotný signál od modulů na svůj výstup. Když jeden z aktivních modulů selže, rozhodovací obvod zamaskuje selhání zvolením většinového signálu jako

správný výstup. Avšak modul, který produkuje rozdílný signál, je označen rozdílovým detektorem a tento modul je spínací jednotkou odpojen z rozhodování. Na jeho místo je zapojen jeden ze záložních modulů. Rozhodovací uspořádání proto dává toleranci pro první chybu v systému. Chybný modul je potom odstraněn k obnovení schopnosti chybové tolerance systému. Množství chybných modulů, které může být vyměněno, závisí na množství poskytnutých náhradních modulů.

Hybridní techniky vyžadují úroveň redundance mezi redundancí statických systémů a redundancí dynamických systémů, a v mnoha případech nabízejí lákavý kompromis, ačkoliv je důležité klást důraz při návrhu spínače. Použití většinového rozhodování maskuje chyby z okolního prostředí, zatímco identifikace chyb a rekonfigurace systému zajišťuje toleranci chyb a provádí údržbu.

Synchronizace modulů a diverzita v hardwarové redundanci

Mnoho aspektů hardwarové redundance je přidruženo k porovnávání mezi provozem rozdílných modulů provádějících stejné provozní operace na hypoteticky stejných vstupních signálech. Ve statických technikách se toto porovnávání provádí ve formě rozhodování, kde jsou ignorovány moduly, které nesouhlasí s ostatními. V dynamickém systému je detekce chyb často provedena porovnáním výstupů, nebo vnitřních signálů, nebo rozdílných modulů. V každém případě je porovnání mnohem snazší provést, pokud může být dosaženo toho, aby signály přidružené ke každému modulu byly vždy identické. V systémech založených na mikroprocesorech je nutné, aby procesory pracovaly v přesné synchronizaci. V praxi může být toto splněno řízením procesorů stejným hodinovým signálem. Za těchto podmínek jsou tato zařízení označována že jsou v **pevném taktu**. Jeden problém s tímto uspořádáním je, že vyžaduje jednu část obvodu – s hodinovým taktem – který je společný pro všechny moduly. To způsobuje tento obvod možným místem pro selhání jednoho bodu. V mnoha případech je nemožné provozovat všechny moduly v pevném taktu. Za těchto podmínek musejí rozhodovací nebo porovnávací metody umožnit časovací rozdíly které nevyhnutelně nastanou mezi moduly.

V předchozí části jsme se podívali na použití diverzity návrhů v boji proti vlivům systematických chyb, jako třeba chyb v návrhu hardwaru, softwaru nebo specifikaci systému. Ačkoliv diverzita nabízí zisk v termínech zvýšené tolerance chyb, může způsobit zajištění redundance složitější. Moduly rozdílných návrhů se často liší v době odezvy, a taky mohou produkovat dočasné rozdíly výstupního signálu, i když pracují správně. K překonání těchto problémů musí být rozhodování nebo porovnávání signálů synchronizováno s provozem systému tak, že všechny moduly produkují identické signály.

4.5 Tolerance softwarových chyb

Termín „tolerance softwarových chyb“ může znamenat dvě rozdílné věci. Nejprve to může znamenat tolerance chyb v softwaru. Použitím této definice můžeme zahrnout do této oblasti několik hardwarových technik probraných v minulé kapitole. Například použití diverzní redundance poskytne ochranu proti systematickým chybám v softwaru stejně tak jako v hardwaru.

Druhá interpretace může znamenat tolerance chyb (jakékoliv formy) použitím softwaru. V této kapitole se zaměříme na techniky spadající do této definice. Některé z těchto technik jsou použity k toleranci chyb v softwaru (a tím vlastně splní i první definici), zatímco jiné mohou být použity k vypořádání se s hardwarovými i se softwarovými chybami.

Ve všech počítačových systémech je software nezbytnou součástí a běžně reprezentuje většinovou část jejich celkové složitosti. Zkušenosti ukazují, že proces vytváření softwaru je

velice náchylný k chybám a je všeobecně akceptováno, že všechny programy jakýchkoliv rozměrů mají chyby, nebo „mouchy“. Byli vyvinuty různé techniky k zabránění zavádění softwarových chyb a k vysledování a vymazání těch, co obcházejí tyto metody. Navzdory těmto technikám je v kritických aplikacích při návrhu systému nezbytná tolerance těchto softwarových much a dalších systematických chyb, protože jejich kompletní odstranění nemůže být zaručeno.

Když je hardwarový modul obsahující procesor duplikován k vytvoření redundance, všechny programy v modulu musejí být taky duplikovány. Když je software v každém modulu identický, pak tato duplikace softwaru nedělá nic ke zvýšení schopnosti systému tolerovat chyby v tomto softwaru – jakákoliv chyba v jedné verzi programu bude přítomna i ve všech kopiích. Duplikace hardwaru poskytuje ochranu proti náhodnému selhání součástky, protože jednotky selhávají v rozdílném čase. Problém v softwaru většinou zasáhne všechny identické moduly ve stejnou chvíli. Ve snaze ochránit systém od softwarových chyb musí být diverzita i v softwaru. Ukážeme si dvě nejpoužívanější metody dosažení tolerance chyb použitím softwaru. To je metoda „programování *N*-verzí“ a metoda „bloků zotavení“.

4.5.1 Programování *N* – verzí

Jak se dá odvodit z názvu, tato technika zahrnuje použití několika rozdílných ztvárnění programu (Chen and Avizienis, 1978; Avizienis, 1985) [2]. Všechny tyto verze se snaží o implementaci stejné specifikace, a proto by měli produkovat stejné výsledky. Rozdílné verze mohou běžet sekvenčně na jednom procesoru, nebo paralelně na více procesorech. Rozdílné postupy používají stejná vstupní data a jejich výsledky jsou porovnány. Při shodě mezi softwarovými moduly je jednoznačný výsledek podstoupen místu určení. Když moduly produkují rozdílné výsledky, pak reakce závisí na počtu použitých verzí. Pro zdvojený systém ($N = 2$) je situace analogická se situací u párů s automatickou kontrolou. Rozdíly mezi moduly označují přítomnost chyby, ale systém nemůže určit, který modul je chybný. Tento problém může být vyřešen opakování výpočtu v naději, že je chyba přechodná. Tento postup může být úspěšný pokud byla chyba způsobena přechodným selháním hardwaru, které narušilo běh procesoru během výkonu softwarového procesu. Nebo se může systém pokusit provést nějakou další diagnostiku k rozhodnutí, který signál je špatný. Lepší uspořádání používá tři nebo více verzí softwaru. Jestliže $N = 3$ nebo více, pak je možné použít nějakou formu rozhodování k maskování vlivu chyby. Je zřejmé, že takové uspořádání je softwarovým ekvivalentem trojité nebo *N* – modulární hardwarové redundance. Ačkoliv velké hodnoty *N* jsou přitažlivé z funkčního hlediska, vysoké náklady je většinou dělají nepraktické.
V reálných aplikacích je velice neobvyklé použití hodnoty *N* větší než 2.

Díky použití diverzity poskytuje programování *N* – verzí rozumnou míru ochrany proti systematickým chybám v softwaru. Nicméně by mělo být pamatováno, že diverzní implementace nezaručují odstranění běžných chyb. Limity diverzity návrhů byli probrány dříve.

Hlavní nevýhodou programování *N* – verzí jsou nároky na výkon procesoru a na cenu provedení. Jestliže je použit jeden procesor, pak se doba výpočtu zvýší o faktor větší než *N* v porovnání s provedením jedné verze. Zvýšení o faktor větší než *N* je způsobeno dodatečnými výpočty způsobenými rozhodovacím procesem. Jestliže je použito *N* procesorů, tento režijní čas není třeba, za cenu dodatečného hardwaru. V druhém případě, náklady na vývoj softwaru se mohou zvýšit o faktor větší než *N*, díky ceně implementace modulů a rozhodovacího softwaru. Tyto vysoké náklady na vývoj omezují použití této techniky ve velice kritických aplikacích, kde může být vysoká cena přijatelná. Příkladem takové aplikace může být základní systém kontroly letu u letadla Airbus A330/340.

4.5.2 Bloky zotavení

Metoda bloků zotavení byla představena začátkem sedmdesátých let dvacátého století (Horning, 1974; Anderson and Lee, 1990) [2]. Používá nějaký druh detekce chyb pro uznání provozu softwarového modulu. Pokud je detekována chyba, pak se použije alternativní část softwarového kódu.

Metoda bloků zotavení je založena na použití **testů akceptace**. Tyto testy můžou mít několik částí a mohou zahrnovat například kontrolu doby běhu programu (například aritmetické chyby, jako dělení nulou), rozumnosti (pro potvrzení, že dostáváme rozumné výsledky), nadměrné doby vykonání příkazu a matematických chyb. Tahle poslední kategorie může zahrnovat obrácení výpočtů pro kontrolu jeho platnosti. Například postup výpočtu základu druhé mocniny může být zkontrolován vynásobením výsledku k ověření že dostaneme původní hodnotu. V kritických aplikacích je normální rozdělit software do zvládnutelných modulů a přesně specifikovat provoz každého modulu. Během vývoje softwaru je nezbytné demonstrovat že každý modul dosahuje funkčnosti zadané ve specifikaci. Takový postup může být také použit k návrhu testů doby běhu programu, čímž můžeme potvrdit, že modul fungoval správně.

Systémy používající bloky zotavení vyžadují duplikaci různých kritických modulů softwaru. V každém případě je vykonám **primární modul**, následován testem akceptace. Selhání testu způsobí vykonání alternativního redundantního modulu, po němž bude test akceptace zopakován.

primární modul
test akceptace
sekundární modul
test akceptace

Může být poskytnut jakýkoliv počet redundantních modulů pro zvýšení tolerance chyb. Jakmile je vykonání jedné z verzí modulu zakončeno úspěšným testem, program pokračuje další softwarovou operací. Pokud test akceptace selže pro všechny redundantní moduly, je detekována celková chyba softwaru a systém musí provést příslušnou akci.

Jeden problém s tímto uspořádáním je, že vykonání chybného modulu může způsobit poškození stavu systému, které musí být opraveno dříve než je obnoven správný provoz. To je vyřešeno vytvořením bodu zotavení před vykonáním modulu, do kterého se systém může vrátit v případě selhání modulu. Lze si tuto operaci představit jako vyfocení stavu všech proměnných v systému. Když je chyba detekována v pozdějším stavu, všechny proměnné mohou být resetovány na tyto hodnoty k odstranění vlivu chyby. Tento proces je příkladem zpětného zotavení chyby. V praxi je pořizování kompletního záznamu celého stavu systému na začátku každého modulu neefektivní, proto jsou použity mnohem důmyslnější techniky. Příkladem je technika obnovení části paměti, která ukládá jen ty hodnoty, které se budou měnit.

Struktura mechanismu bloku zotavení se proto skládá z:

vytvoření bodu obnovení
primární modul
test akceptace
alternativní modul 1
test akceptace
alternativní modul 2
.
.
.
alternativní modul n
test akceptace

Alternativní metoda popisu této struktury používá malé množství klíčových slov pro vyjádření různých stavů procesu (Anderson and Lee, 1990) [2]. Test akceptace je identifikován slovem **ensure**. To je umístěno na začátek struktury a je běžné pro všechny verze modulu. Pokračuje primární modul, identifikován slovem **by**, následován jakýmkoliv počtem alternativních modulů, každému předchází slovo **else by**. Modul je ukončen větou **else error**, oznamující událost, že všechny moduly selhali při testu akceptace.

ensure *(test akceptace)*
by *(primární modul)*
else by *(alternativní modul 1)*
else by *(alternativní modul 2)*
.
.
.
else by *(alternativní modul n)*
else error

Tento syntax neindikuje vytvoření bodu obnovení, ale tato operace je zahrnuta na začátku každého bloku.

4.5.3 Porovnání hardwarových a softwarových technik tolerance chyb

Je zřejmé, že tu je veliká míra shody mezi metodami softwarové tolerance chyb a hardwarovými technikami. Programování *N* – verzí poskytuje maskování chyb stejným způsobem jako uspořádání *N* – modulové redundance, porovnávání výsledků v prvním bývá ekvivalentní rozhodovacímu systému ve druhém. Bloky obnovení používají detekci chyb k přepínání mezi primárním a záložními softwarovými moduly, a je to podobné technikám použitým ve schématech dynamické hardwarové redundance.

Všechny formy hardwarové tolerance chyb vyžadují poskytnutí prostředků hardwarové redundance. Softwarová redundance také vyžaduje zvýšení zdrojů k dosažení stejné funkčnosti. V případě programování *N* – verzí musí být provedeno několik verzí modulu, s dodatečnými režijními prostředky pro rozhodovací operace. V schématu bloků obnovení jsou duplikované kalkulace vyžadovány pouze když modul selže, ale režie při testování akceptace je se vší pravděpodobností znatelně větší, než režie při rozhodování. V aplikacích kde je časování kritické, je nezbytné, aby byl k dispozici čas pro provedení ne jenom primárního, ale i alternativních modulů.

Hlavní rozdíl mezi hardwarovými a softwarovými technikami chybové tolerance je, že duplikace identických hardwarových modulů může být použita k poskytnutí vyšší tolerance nějakých druhů hardwarových chyb, zatímco duplikace identických softwarových modulů má jen malý účinek. Opakované vykonání identických softwarových modulů může poskytnout ochranu proti přechodným chybám, ale neposkytuje ochranu proti chybám v návrhu softwaru. Softwarová redundance je proto stále sdružena s diverzním návrhem softwarových modulů. Kvůli velikým nákladům tohoto procesu je tento druh chybové tolerance většinou použit pouze ve vysoce kritických aplikacích.

4.5.4 Výběr architektury tolerance chyb

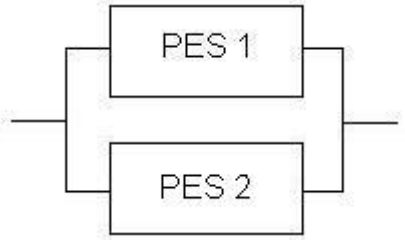
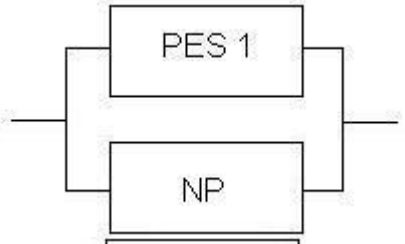
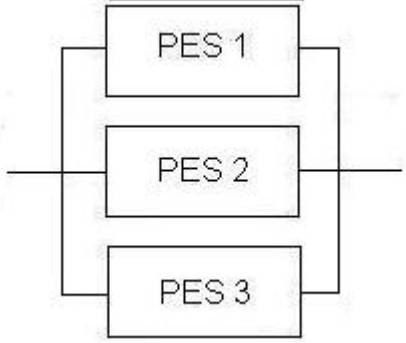
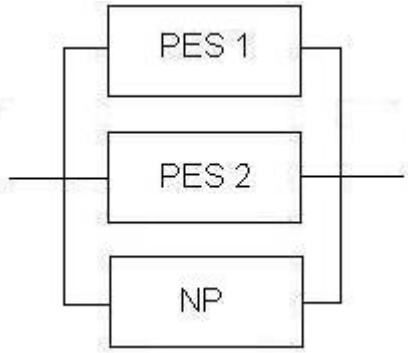
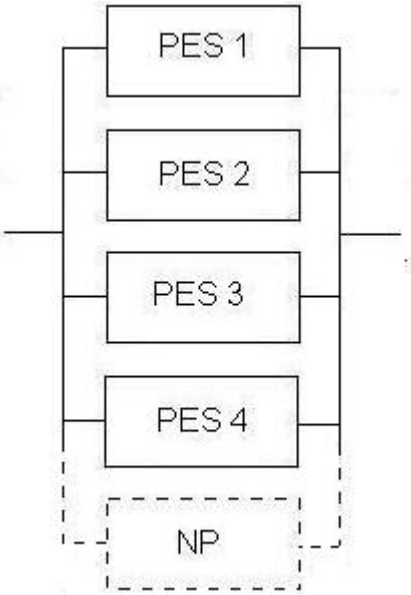
Probrali jsme velké množství systémů tolerance chyb s různými stupni složitosti. Výběr příslušné architektury pro danou aplikaci vyžaduje dovednost a zkušenosti. Často může být získáno mnoho poučení ze standardů a směrnic použitých v daném průmyslovém odvětví.

V této práci se hlavně zaměřujeme na systémy založené na počítačích, a v této kapitole jsme se zaměřili na uspořádání programovatelných modulů. V některých aplikacích může být možné provést některé aspekty systému použitím neprogramovatelných prvků. Složitost je velkým nepřítelem bezpečnosti, a v systémech s důrazem na bezpečnost je jí pokud možno zabráněno. Proto pokud lze funkcí modulu dosáhnout použitím neprogramovatelných prvků, je to se vší pravděpodobností preferováno v bezpečnostních aplikacích.

V mnoha kontrolních systémech jsou požadované funkce příliš složité, aby mohli být implementovány použitím neprogramovatelných technologií. Nicméně může být možné oddělit kontrolní a bezpečnostní aspekty systému, čímž i relativně jednoduchý neprogramovatelný prvek může zaručit bezpečnost systému.

Naneštěstí není vždy možné oddělit kontrolní a bezpečnostní funkce systému. V systému leteckého autopilota je například nepraktické vytvořit jednoduchý neprogramovatelný systém, který by zajistil bezpečnost letadla pokud počítačový systém selže. Nicméně kde je to možné, je vždy preferováno zajištění bezpečnosti použitím jednoduchých systémů před spoléháním se na z podstaty složité počítačové systémy.

Všechny systémy tolerance chyb, popsané v této kapitole, používají redundanci k zabránění selhání subsystému a tím i selhání celého systému. V některých aplikacích může být ztráta kontrolní funkce přijatelně vyřešena tak, že bezpečnost systému je zajištěna. Toto může být dosažitelné použitím uspořádání ve kterém jsou některé redundantní kanály neprogramovatelné. Tab. 1 ukazuje několik příkladů systémů s tolerancí chyb. V této tabulce je počítačovým modulům dána zkratka PES, která znamená programovatelný elektronický systém – toto značení je široce požívané v bezpečnostních aplikacích. Nepočítačové moduly jsou označeny jako NP – neprogramovatelné. Tab. 1 se nesnaží určit vodítka pro výběr architektury s tolerancí chyb pro konkrétní aplikaci. Spíše ukazuje několik příkladů, jaké druhy systému mohou být na různých úrovních integrity. Tabulka nenaznačuje detaily architektury – například neurčuje, zda je použit statický nebo dynamický systém – jednoduše určuje příslušnou úroveň redundance. Jak je vidět, tak neprogramovatelné kanály často nahrazují počítačové moduly tam, kde je to možné.

	Dva kanály, oba programovatelné Kanály mohou být stejné nebo rozdílné	Vhodné pro méně kritické aplikace Automobilové, železniční a některé letecké systémy
	Dva kanály, jeden programovatelný, druhý neprogramovatelný	Vhodné pro méně kritické aplikace Automobilové, železniční a některé letecké systémy
	Tři kanály, všechny programovatelné Všechny kanály identické	Vhodné pro vysoce kritické aplikace Kontrola riskantních procesů a kritické letecké systémy (jako systém automatického přistávání)
	Tři kanály, dva programovatelné, a jeden neprogramovatelný Programovatelné kanály identické	Vhodné pro vysoce kritické aplikace Kontrola riskantních procesů a kritické letecké systémy (jako systém automatického přistávání)
	Čtyři programovatelné kanály Mohou být čtyři identické kanály a dva diverzní páry Může používat další neprogramovatelný ochranný kanál	Vhodné pro nejkritičtější aplikace Nukleární reaktory a systémy letecké kontroly

Tab. 1: Příklady uspořádání systémů s tolerancí chyb

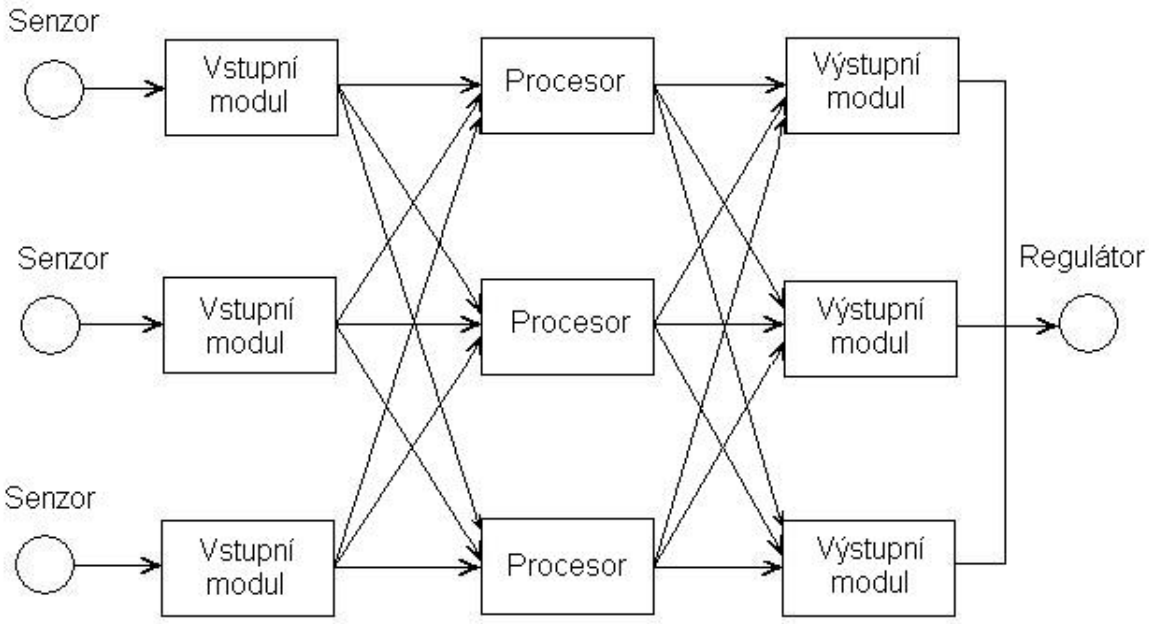
4.6 Příklad systému s tolerancí chyb

Programovatelné logické kontroléry s tolerancí chyb

Programovatelné logické kontroléry (PLC) jsou stále častěji používány jako základ kontrolních systémů malého rozsahu. To jsou masově produkované embedded systémy, které jsou dostupné s rozsahem vstupních a výstupních modulů a vyhovují rozsahu aplikací. Původně byli PLC zamýšleny jako jednoduchá náhrada za existující neprogramovatelný kontrolní hardware, a také jsou často použity k nahrazení systémů založených na velkém množství relé. Z toho důvodu byli první PLC obvykle programovány použitím „příčkové logiky“, diagramové programovací techniky často použité k návrhu reléových obvodů.Dnes je k dispozici množství programovacích technik pro PLC, včetně použití více konvenčních programovacích jazyků.

Během několika posledních let bylo věnováno mnoho pozornosti použití PLC v bezpečnostních aplikacích (Goring, 1994; Greenway, 1994)[2]. Některé PLC používají techniky tolerance chyb přímo v návrhu pro zvýšení jejich spolehlivosti. Typická jednotka by používala tři identické procesory v uspořádání TMR, se zdvojenými nebo ztrojenými vstupními a výstupními obvody. Takové uspořádání je zobrazeno na Obr. 10.

Konfigurace TMR poskytuje toleranci a maskování náhodných selhání hardwaru v procesoru a přidružených součástí. Toto, společně se zabudovanými systémy detekce chyb, vytvoří systém který je znatelně spolehlivější než srovnatelný návrh používající jeden procesor. Nicméně použití identických hardwarových modulů, každý se stejným softwarem, neposkytuje ochranu proti systematickým chybám.



Obr. 10: Typické uspořádání vysoce spolehlivého PLC

5 Analýza rizika

Pravděpodobně tím nejdůležitějším mechanismem pro zvýšení bezpečnosti systému je rozpoznání událostí, které mohou způsobit poškození. Jakmile jsou tyto problémové oblasti lokalizovány, může být stanovena jejich důležitost, a pokud jsou shledány nebezpečnými, mohou být podniknuty patřičné kroky k jejich odstranění nebo zmírnění jejich efektů.

Situace, které mohou způsobit poškození jsou nazývány **rizikem**, což je situace, při které hrozí nebezpečí lidem nebo okolnímu prostředí.

V této kapitole se podíváme na několik metod analýzy rizika v systému.

5.1 Analytické techniky

Obecný postup při analýze rizika

a) popis rozsahu a cíle analýzy,
b) stanovení potenciálních nebezpečí (zdrojů rizika),
c) kvantitativní vyhodnocení pravděpodobnosti nebo četnosti havárií,
d) kvantitativní vyhodnocení následků havárií (ztrát či zranění),
e) souhrn informací získaných zobrazením rizik,
f) stanovení, která rizika jsou přijatelná nebo přípustná,
g) úprava nebo zdokonalení konstrukce zařízení a pracovního postupu s cílem snížit pravděpodobnost projevu rizika,
h) zajištění realizace přijatých opatření.

Analýzu rizika u již provozovaných technologií je třeba provést při změně způsobu skladování a dopravy v provozu, při výstavbě nových provozů nebo při modernizaci technologických procesů, při změně typu výroby v provozním zařízení, při změně způsobu řízení procesu, např. zavedení automatizace nebo při podstatných personálních změnách, při provádění závažnější údržby nebo při podstatnějších opravách v provozu,

Dále se při analýze rizik můžeme setkat s těmito pojmy:
a) kvantitativní analýza rizika - odhad četnosti nebo pravděpodobnosti havárie,
b) kvalitativní analýza rizika odhaduje nebo vypočítává následky havárií,
c) termín riziko zahrnuje dva vyhodnocované faktory: následky a pravděpodobnost.

Definice druhů rizik

a) individuální riziko je riziko, kterému je vystaven jedinec v dosahu rizika,
b) skupinové riziko je definováno jako riziko skupiny osob, např. týmu. Míra skupinového rizika pro jednotlivé osoby souvisí s počtem osob ve skupině a průměrným individuálním rizikem,
c) sociální riziko je riziko pro společnost jako takovou. Může být odhadnuto pomocí četnosti závažných havárií. Je tím vyšší, čím více osob může být projevem rizika postiženo,
d) závažné nebezpečí je definováno pro praktické účely jako možná příčina havárií, při kterých může být usmrceno 10 a více osob, zraněno více než 100 osob, materiálové ztráty nebo ztráty produkce jsou vyšší než 10 mil. USD nebo může dojít k závažnému znehodnocení životního prostředí[2].

Objektivní identifikace nebezpečí má poskytnout seznam pravděpodobných poruch nebo jejich kombinací vedoucích k haváriím. K tomuto účelu můžeme použít řady rozdílných metod lišících se:
a) v podkladech potřebných pro provedení,
b) podle počátečního stádia,
c) ve způsobu provedení v závislosti na čase,
d) ve způsobech kombinace možných poruch.

Faktory ovlivňující výběr vhodné metody

a) cíl metody (druh požadovaných výsledků). Nejčastěji se jedná o vytvoření seznamu nebezpečných stavů, o návrh úprav vedoucích ke zvýšení bezpečnosti, o seznam závažných nebezpečí a o posouzení rizika,
b) typ analýzy (zda se jedná o první, opakovanou či speciální analýzu),
c) informace potřebné k provedení analýzy, především její dostupnost,
d) charakteristiky analyzovaného procesu - složitost procesu, charakter procesu, typy procesních operací, charakter vlastního nebezpečí látky,
e) zkušenosti s používaným procesem - rozsah dosavadních zkušeností s haváriemi a řešením havarijních stavů (též studium podobného procesu),
f) náklady na analýzu - jedná-li se o provoz, který je charakterizován dlouhodobým obdobím bezporuchového provozu, lze použít pro analýzu metody, která je jednodušší, méně systematická, ale i méně nákladná.

Při analýze rizika se nepoužívá jedna, ale množství technik, každá poskytující rozdílný pohled do charakteristik zkoumaného systému. Některé metody byli vyvinuty v konkrétních odvětvích průmyslu a mají omezené využití v jiných oblastech. Jiné, ačkoliv mají původ ve specifických oblastech, byli shledány široce použitelnými v mnoha průmyslových oblastech.

Těmi nejvíce používanými technikami jsou:

- tradiční metody,
  a) Check List Analysis - analýza pomocí kontrolních záznamů,
  b) Safety Audit - bezpečnostní audit,
  c) What if... - co se stane, když...,
- metody relativního hodnocení (Relative Ranking)
- úvodní analýzy nebezpečí (Preliminary Hazard Analysis)
- studie nebezpečí a provozuschopnosti (Hazard and Operability Studies - HAZOP)
- analýza poruch a jejich následků (Failure Modes and Effects Analysis - FMEA)
- analýza softwarových poruch a jejich následků (Software Failure modes and Effect Analysis - SFMEA)
- analýza poruch, následků a kritičnosti selhání – (Failure Modes, Effect and Criticality Analysis - FMECA)
- analýza příčin následků (Cause Consequence Analysis),
- analýza spolehlivosti člověka (Human Reliability Analysis - HRA)
- výzkum rizika a provozuschopnosti (Hazard and Operability Studies - HAZOP)
- analýza stromem událostí (Event Tree Analysis - ETA)
- analýza stromem chyb (Fault Tree Analysis - FTA)
- analýza stromem softwarových chyb (Software Fault Tree Analysis - SFTA)
- analýza softwarového rizika (Software Risk Analysis - SRA)

Pro porovnání si některé z metod podrobněji popíšeme.

What if...? identifikuje nebezpečné stavy v technologickém procesu. Pomocí charakteristických otázek, začínajících slovy "Co se stane, když..." jsou udávány možné příčiny havárií a navrhují se opatření pro zvýšení bezpečnosti. Může být vznesena jakákoliv námitka týkající se bezpečnosti procesu.

Sestavování charakteristických otázek, směřujících k identifikaci nebezpečí, však není v tomto případě systematizováno, jako je tomu např. u metody HAZOP. Kladení otázek a odpovědí závisí na zkušenostech a intuici týmu odborníků, který studii uskutečňuje. Probíhá formou porad vybraných odborníků podrobně seznámených s procesem. Při poradách se důsledně uplatňuje brainstorming - spontánní diskuse o hledání nových nápadů. Metoda je velmi účinná, pokud studii provádí tým vysoce kvalifikovaných odborníků.

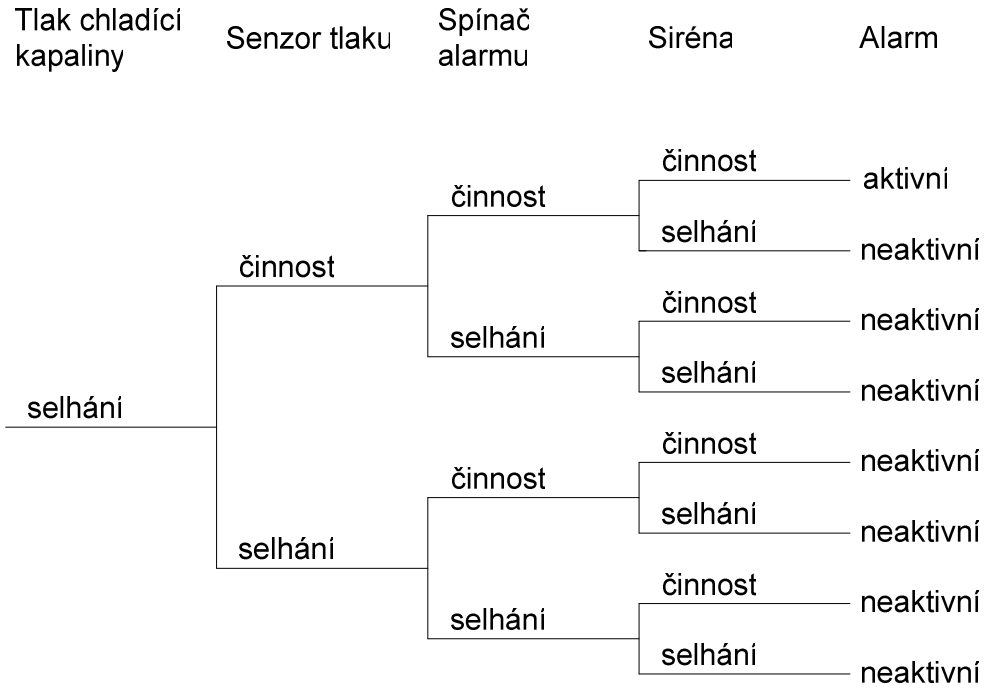
Failure modes and effect analysis (FMEA) uvažuje selhání jakékoliv součásti v systému a sleduje vliv těchto selhání k určení jejich konečných následků. Tento proces vytváří předpoklady selhání součástek a pak určuje efekt těchto selhání na celý systém. Analýza může být provedena na hardwarové úrovni součástek, nebo na provozní úrovni použitím modulového přístupu. Protože uvažuje selhání všech komponent, je tento přístup částečně vhodný při detekci podmínek, kde i jedno selhání může vést k nebezpečné situaci. Avšak tyto techniky většinou neuvažují vícenásobná selhání. Bohužel, protože analýzy pohlížejí na vliv selhání všech součástek, mnoho práce je asociováno se selháními, které nevyústí v nebezpečné podmínky. FMEA zahrnuje mnohem detailnější, náročnější práci a proto je drahá její úplná aplikace na rozsáhlé složité systémy. Často je FMEA použita v pozdějším stádiu vývojového procesu, kdy je aplikována spíše na kritická místa než na celý systém.

Failure modes, effects and criticality analysis (FMECA) je rozšířením FMEA, která navíc bere v úvahu důležitost selhání každé součástky. Toho je dosaženo uvažováním následků konkrétních selhání, a jejich pravděpodobnosti nebo frekvence jejich výskytu za účelem identifikace těch sekcí systému, kde jsou selhání nejnebezpečnější. To umožňuje naše úsilí nasměrovat tam, kde je nejvíce potřeba.

Hazard and operability studies (HAZOP) používá sérii takzvaných „pomocných slov“ k vyšetření vlivů odchylek od normálních provozních podmínek během každé fáze provozu systému. HAZOP byl původně vyvinut v chemickém průmyslu a je konkrétně výkonný při demonstrování vlivů parametrických změn a hodnot mimo rozsah na bezpečnost. Zkoumání spoléhá na určení odpovědí na otázky typu „co když“, například: „Co způsobí zvýšení teploty?“ nebo „ Co se stane, když se sníží tlak oleje?“ Ačkoliv HAZOP může být plně v režii počítače, zkoumání se silnou měrou spoléhá na údaje získané od expertů. Celý proces může být velice efektivní, ale také velice únavný a časově náročný.

Event tree analysys (ETA) si jako počáteční bod bere události, které mohou ovlivnit činnost systému a sleduje jejich průběh pro určení možných následků. Mezi ně se řadí běžné události vznikající činností systému, jako otevření nebo zavření ventilu, a chybové události, jako třeba selhání některé z komponent. Pro složité systémy jsou výsledkem velmi rozvětvené stromy událostí, když uvažujeme za nebezpečné i normální operace systému. Základní formou stromu událostí může být příklad na Obr.11. Ten zobrazuje strom událostí pro malý podsystém ovládající výstražné zařízení. Zde je použit tlakový senzor pro detekci dostatečného tlaku chladící kapaliny zajišťující bezpečný provoz chemického závodu. Pokud tlak poklesne pod určitou přednastavenou hodnotu, senzor by tento pokles měl detekovat, a aktivovat vypínač alarmu, čímž se spustí výstražná siréna. Strom událostí začíná na levé

straně diagramu událostí odpovídající poklesem tlaku chladící kapaliny. Strom se poté rozděluje do dvou možných následných událostí. První událostí je správná činnost senzoru tlaku,což zobrazuje horní větev, při druhé, zobrazené dolní větví, dojde k selhání senzoru. Pro každou z těchto možností nyní uvážíme možnou akci dalšího článku v řetězci, spínače alarmu. V obou případech může pracovat správně, nebo může selhat, což vede ke čtyřem možným cestám. Nakonec je nutné uvažovat činnost sirény pro každou z těchto cest. Protože každá z událostí vytváří novou větev diagramu, strom s *N* událostmi bude mít 2^N větví. Na pravé straně diagramu vidíme možné reakce systému na původní událost. Můžeme vidět, že když všechny prvky v systému fungují korektně, siréna bude znít, ale pokud jeden z prvků selže, k alarmu nedojde. Možná že na tomto jednoduchém příkladu není tento výsledek moc překvapující. Síla analýzy pomocí stromu událostí je v tom, že u mnohem složitějších uspořádání dovoluje podrobnější prošetření vlivů událostí i v situacích, kdy jejich následky jsou mnohem méně zřejmé.



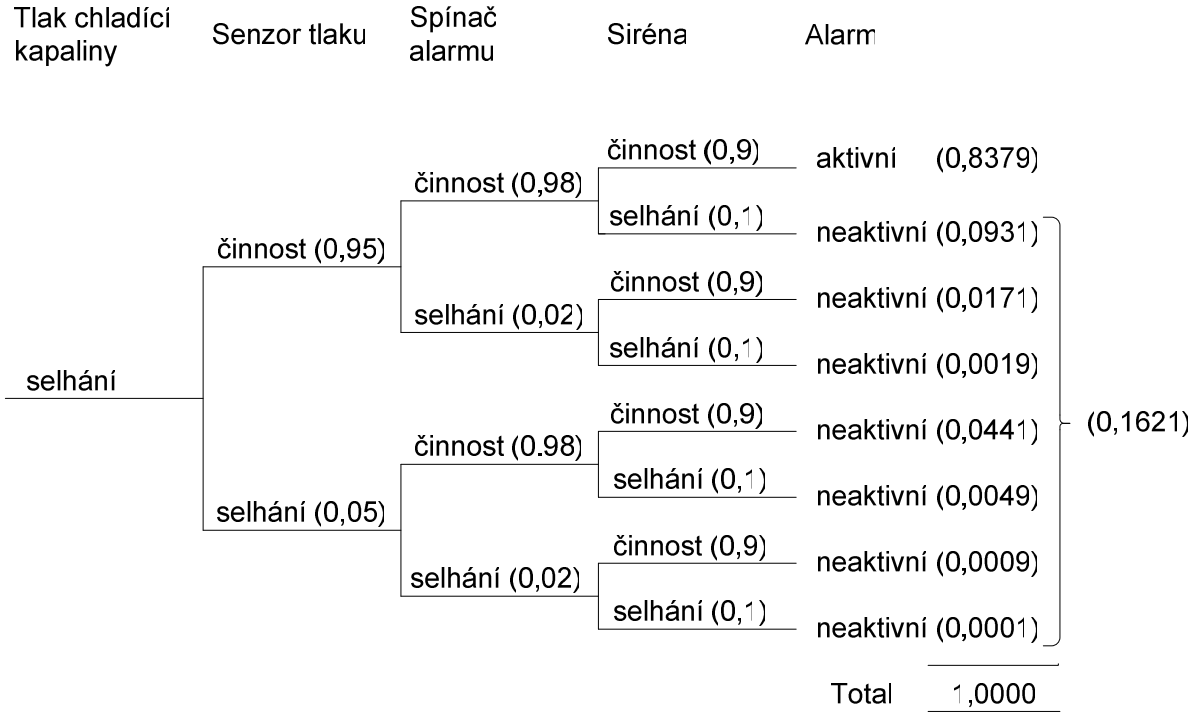
Obr 11: Jednoduchý strom událostí

Fault tree analysis (FTA) se odlišuje od použití stromu událostí v tom, že nahlíží na problém v opačném směru. Strom událostí začíná se všemi možnými událostmi a pracuje dopředu k určení jejich následků. Následkem toho se většina analýz týká událostí, které nemají žádné bezpečnostní důsledky. Naproti tomu analýza pomocí stromu chyb začíná se všemi identifikovanými riziky a pracuje zpětně k určení jejich pravděpodobných příčin. V aplikacích, kde jsou dostatečné informace dostupné z provozu podobných systémů, mohou být jako počáteční bod analýzy také použity data z předchozích nehod nebo incidentů. Ke zjištění vztahů mezi příčinami a následky událostí jsou použity logické operátory podobné těm z boolenovské algebry. Zaměření se na událostí, které jsou známými příčinami rizik vyústí v jednodušší strukturu stromu než je tomu u analýzy pomocí stromu událostí.

5.2 Pravděpodobnostní analýza rizik

Ačkoliv se tyto techniky analýzy chyb primárně zabývají spíše identifikací rizik než jejich počítáním, je nutné zmínit, že několik z nich je možné použít i k získání nějaké míry četnosti výskytu nebo pravděpodobnosti chyby. To je nazýváno výrazem **pravděpodobnostní analýza rizik**, a tvoří podstatnou část celkového procesu analýzy rizik. Tento přístup je běžně používán spolu s analýzou FMECA, kde je měřítko kritičnosti použito na každý možný způsob selhání. To dovoluje analýze koncentrovat se na oblasti s největší důležitostí. Strom událostí a technika stromu chyb mohou být také použité k poskytnutí pravděpodobnostní informace. Toho je docíleno přiřazením pravděpodobností ke každé větvi podle odpovídající pravděpodobnosti chyby jdoucí touto cestou. Kombinací pravděpodobností jednotlivých větví můžeme získat celkovou pravděpodobnost selhání. Příkladem použití pravděpodobnostních metod v analýze stromem událostí je Obr. 12 , vycházející z předchozího obrázku.

Základní struktura stromu na Obr. 12 je shodná s předchozím obrázkem, až na to, že teď byli pravděpodobnosti přidány ke každé větvi. Je například zřejmé, že tlakový senzor má pravděpodobnost správné funkce 0.95 a pravděpodobnost selhání 0.05. Všimněte si, že součet pravděpodobností je na každé větvi jednotný, protože jedna z možných událostí musí nastat. Pravděpodobnost u každé větve na pravé straně je jednoduše součtem pravděpodobností podél této cesty. Takovýmto sčítáním pravděpodobností lze jednoduše určit výslednou pravděpodobnost konkrétního selhání. V tomto příkladu je pravděpodobnost správné funkčnosti přibližně 0.84 a pravděpodobnost selhání 0.16. a opět musí tyto pravděpodobnosti být v součtu jedna [2].



Obr 12: Pravděpodobnostní analýza aplikovaná na strom událostí

Ačkoliv možnost výpočtu pravděpodobnosti konkrétní chyby je velice lákavá, musí být pamatováno, že výsledná hodnota je platná, pouze pokud jsou platné i hodnoty v stromu událostí. V některých případech mohou být pravděpodobnosti vzniku jednotlivých větví získány ze starých statistických dat. V jiných případech může být nezbytné určit

pravděpodobnosti na základě inženýrských posudků. Při vyhodnocování důležitosti jednotlivých hodnot musí být každý předpoklad pečlivě prozkoumán.

5.3 Failure modes and effect analysis (FMEA)

Failure mode and effect analysis (FMEA) je procedura pro analýzu potenciálních módů selhání v systému pro klasifikaci podle kritičnosti nebo pro určení efektů selhání na systém. Je široce použita ve výrobním průmyslu v různých fázích výrobního cyklu. Selhání jsou způsobena chybami nebo defekty v procesu, návrhu nebo při používání, a mohou být skutečná nebo jen potenciální. Effect analysys se snaží studovat následky těchto selhání.

Základní termíny:

Způsob selhání - Způsob, jakým byla chyba zpozorována, většinou popisuje způsob vzniku chyby.
Vliv selhání – Okamžité následky selhání na provoz, funkci nebo stav systému
Lokální následky – Vliv selhání tak, jak je aplikován na jednotku pod analýzou
Následky na větší část – Vliv selhání tak, jak je aplikován na větší část systému
Konečný vliv – Vliv selhání na celý systém
Příčina selhání – Chyby v návrhu, procesu výroby, kvalitě nebo aplikace součástek, tvořící hlavní příčinu selhání nebo spouštějící proces vedoucí k selhání.
Kritičnost – Následky způsobu selhání. Kritičnost zvažuje nejhorší možné následky selhání, určené stupněm zranění, poškození majetku nebo systému, které může nastat.
Certifikační třída – Identifikátor složitosti systému. S přiblížením k jedné s složitost zvyšuje.

Při implementaci FMEA jsou chyby řazeny podle toho, jak vážné jsou jejich následky, jak často nastávají a jak snadno mohou být detekovány. FMEA také dokumentuje současné znalosti o riziku selhání pro použití při soustavném zlepšování. FMEA je použita během stadia vývoje se zaměřením odvrátit možná selhání. Později je použita při kontrole výrobního procesu, před a po výstupních operacích procesu. V ideálním případě FMEA začíná v ranných stádiích návrhu a pokračuje po celou dobu života produktu nebo služby.

Účelem FMEA je podniknou kroky k odstranění nebo snížení počtů selhání. Může být použita k zhodnocení priorit risk managementu pro snížení známých hrozeb. FMEA pomáhá vybrat nápravné akce které snižují kumulativní vlivy následků systémového selhání v životního cyklu.

5.3.1 Použití FMEA při návrhu systému

FMEA může poskytnou analytický přístup při zpracování potenciálních módů selhání a s nimi spojených příčin. Při zvažování možných selhání v návrhu, jako bezpečnost, cena, výkon, kvalita a spolehlivost, může inženýr získat mnoho informací jak pozměnit vývojový nebo výrobní proces ve snaze odvrátit tato selhání. FMEA poskytuje snadný nástroj pro určení, které riziko má největší důležitost, a která akce je nutná k odvrácení problému před tím, než nastane. Vývoj těchto specifikací zajistí, že produkt splní očekávané požadavky.

Proces provádění FMEA je přímočarý. Je vyvinut ve třech hlavních fázích, ve kterých musejí být definovány potřebné akce. Ale před spuštěním FMEA je důležité provést některá opatření pro zajištění, že robustnost a oznaky z minulosti jsou zahrnuty v analýze.

Z počátku je nutné popsat systém a jeho funkce. Dobré pochopení systému usnadňuje další analýzu. Tímto způsobem může inženýr vidět, která použití systému jsou žádaná a která

ne. Je důležité posoudit jak úmyslná tak neúmyslná použití. Neúmyslná použití jsou formou nepřátelského prostředí.

Dále je nutné vytvořit blokový diagram systému. Tento diagram poskytuje přehled hlavních součástek nebo kroků procesu a jejich interakci. To se nazývá logické relace na kterých může být FMEA vytvořena. Je užitečné vytvořit kódovací systém pro identifikaci různých prvků systému.

Před spuštěním funkční FMEA je nutné vytvořit pracovní list, který obsahuje důležité informace o systému, jako datum revize nebo názvy součástek. V tomto pracovním listu by všechny položky nebo funkce subjektu měly být seřazeny v logické posloupnosti, založené na blokovém diagramu.

Kritičnost

Určuje všechny způsoby selhání podle funkčních požadavků a jejich efektů. Příkladem způsobu selhání může být: elektrický zkrat, koroze nebo deformace. Je důležité zmínit, že způsob selhání v jedné součásti může vést ke způsobu selhání v jiné součásti. Proto by každý mód selhání měl být řazen v technických termínech a podle funkce. Od tohoto okamžiku musí být uvažován konečný efekt každého selhání. Vliv selhání je definován jako výsledek selhání na funkci systému jak je vnímána uživatelem. Tímto způsobem je vhodné zapsat tyto vlivy v termínech, co uživatel může vidět nebo zažít. Příklady vlivu selhání jsou: snížený výkon, hluk nebo dokonce zranění uživatele. Každý vliv je ohodnocen **stupněm kritičnosti** od 1 (žádné nebezpečí) do 10 (vysoké nebezpečí). Tyto čísla pomáhají inženýrům v určování priorit. Pokud kritičnost vlivu má číslo 9 nebo 10, jsou zvažovány akce ke změně designu odstraněním způsobu selhání, pokud je to možné, nebo ochranou uživatele před vlivem selhání. Kritičnost 9 a 10 je všeobecně rezervována pro ty vlivy, které mohou způsobit zranění uživatele nebo jinak vyústit například v žalobu.

Četnost výskytu

Četnost výskytu je dalším důležitým parametrem FMEA, určuje, jak často může příčina selhání nastat. To může být zjištěno z podobných produktů nebo procesů a selhání, které u nich byli zdokumentovány. Příčina selhání je brána jako slabina v návrhu. Všechny potenciální příčiny selhání by měly být identifikovány a zdokumentovány. A opět by to mělo být provedeno v technických termínech. Příkladem příčin je: chybný algoritmus, nadměrné napětí nebo nesprávné provozní podmínky. Chybovému módu je dáno **pravděpodobnostní číslo**, opět od 1 do 10. Jsou nutné další nápravná opatření, pokud je míra výskytu vysoká (znamená >4 pro nebezpečné způsoby selhání a >1 pokud míra kritičnosti je mez 9 a 10). Tato kroky jsou nazývány detailní vývojovou sekcí procesu FMEA.

Detekce

Jakmile jsou určeny odpovídající opatření, je nutné otestovat jejich účinnost. Také je nutná verifikace návrhu. Potřebné je i určení odpovídajících testovacích metod. Nejprve by se inženýr měl zaměřit na vlastní kontrolu systému, která zamezuje výskytu způsobů selhání nebo která detekuje selhání před tím, než jeho následky ovlivní zákazníka. Poté by měl určit testovací, analytické, monitorovací a další techniky, které mohou být nebo byli použity na podobných systémech k detekci selhání. Z těchto kontrol se může inženýr naučit, s jakou pravděpodobností je možné selhání identifikovat nebo detekovat. Každá kombinace předchozích dvou parametrů určuje **detekční číslo**. Toto číslo představuje schopnost plánovaných testů a prověrek odstranit defekty nebo detekovat způsoby selhání.

Číslo priority rizika

Číslo priority rizika slouží ke stanovení prahových hodnou při určování akcí proti způsobům selhání. Poté, co je určena kritičnost, četnost výskytu a detekce může být číslo priority rizika snadno vypočteno vynásobením těchto tří hodnot.

To musí být provedeno pro celý proces nebo návrh. Jakmile je výpočet hotov, je snazší určit oblasti s nejvyšším výskytem selhání. Způsoby selhání s nejvyšším číslem priority by měly mít nejvyšší prioritu pro nápravná opatření. To znamená, že ne vždy by měl být řešen jako první způsob selhání s nejvyšším číslem kritičnosti. Mohou nastat selhání s nízkou kritičností, ale s vysokou pravděpodobností a nízkou mírou detekce.

Poté, co jsou tyto hodnoty zaměřeny, určí se cíle, odpovědnost a data nápravných opatření. Tyto akce mohou zahrnou konkrétní prověrky, testování a procedury kvality, změny v návrhu (výběr nových komponent), přidání redundance a omezení vlivu prostředí nebo operačního působení. Jakmile jsou tyto akce implementovány do návrhu nebo výrobního procesu, nové číslo priority rizika by mělo být vypočteno pro ověření správnosti úprav. Kdykoliv je návrh nebo proces systému změněn, FMEA by měla být upravena.

Důležité poznatky jsou:

- Pokusit se eliminovat způsob selhání
- Snížit kritičnost selhání
- Snížit četnost výskytu způsobu selhání
- Zvýšit detekci

FMEA by měla být upravena:

- Na začátku nového cyklu (nový produkt, proces)
- Při změnách v operačních podmínkách
- Při změnách v návrhu
- Při ustanovení nových předpisů
- Zpětná vazba od zákazníka značí problém

FMEA postupně vybírá jednotlivé komponenty nebo funkce v systému vyšetřuje jejich možné způsoby selhání. Poté uvažuje možné příčiny každého způsobu selhání a ohodnotí jejich pravděpodobné následky. Účinky selhání jsou určené pro samotnou jednotku a pro celý systém, a jsou navržené možné nápravné akce (IEC, 1982)[2].

Tato technologie může být použita v různých stupních vývoje systému. Často je použita na funkční úrovni v ranném stádiu životního cyklu daného systému, kdy to může být výhodné při zjišťování požadovaného stupně bezpečnostní integrity. Nebo také může být použita ve velmi pozdní fázi vývoje, poté co je většina z návrhových prací hotova. Zde může být aplikována na úrovni součástek i na funkční úrovni. Jednoduchý příklad FMEA je v Tab. 2 .

FMEA pro mikrospínač						
Ref. číslo	Jednotka	Způsob selhání	Možná příčina	Lokální následky	Následky na systém	Nápravné opatření
1 2	Bezpečnostní spínač elektrického zařízení	Přerušené kontakty	(a) chybná součástka (b) nadměrný proud (c) extrémní teplota	Selhání detekce spínače	Zamezuje použití zařízení – systém přepne do chybového stavu	Výběr spínače s vysokou spolehlivostí a nízkou pravděpodobností nebezpečného selhání Přísní kontrola kvality dodaných spínačů
3		Zkratované kontakty	(a) chybná součástka (b) nadměrný proud	Systém chybně detekuje spínač jako sepnutý	Dovoluje použití zařízení i když pojistka je neaktivní – nebezpečné selhání	Úprava software k detekci selhání spínače a provedení příslušných akcí
		Nadměrné kmitání kontaktů spínače	(a) stárnutí materiálu (b) dlouhotrvající vysoký proud	Nepatrné zpoždění v zjišťování stavu spínače	Nepatrný	Hardware navrhnutý tak, aby zamezil nadměrnému proudu skrze spínač

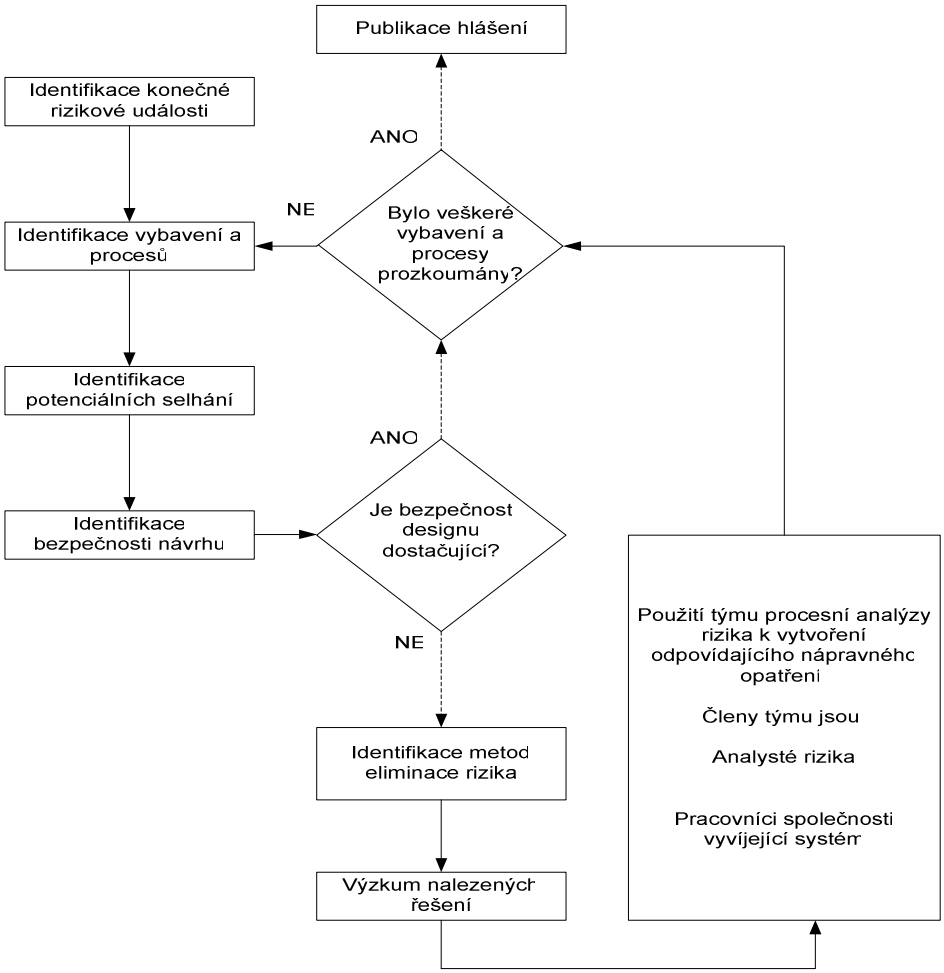
Tab. 2.: Jednoduchá tabulka FMEA

FMEA může být aplikována v několika úrovních pro zdokonalení analýzy. Například předběžná analýza se může zaměřit na účinky selhání motoru na letadlo. To může být následováno sledováním součástek v motoru, například benzinového čerpadla, a pak sledováním součástek v čerpadlu, například ventilu. Provedením této analýzy v týmu čítajícím od čtyř do osmi inženýrů lze získat mnoho užitečných informací [2].

Analýza může zahrnout i pravděpodobnostní část zahrnutím pravděpodobnostních a kritických informací do tabulky. Tak jako u dalších druhů analýzy je její efektivita kontrolována úsudkem a zkušenostmi zúčastněných inženýrů. FMEA je často použita pro poskytnutí vstupních dat pro analýzu pomocí stromu událostí, tyto dvě metody se navzájem doplňují.

Během FMEA analýzy je každý případ vyhodnocen pro adekvátní bezpečnost návrhu a potenciální dopad na systém. Poté je pro každý případ selhání přiřazena kvalitativní kategorie rizika podle daných směrnic. Toto kvalitativní hodnocení je podmíněno uvážením rozsahu a častosti výskytu chyby.

Kritické části procesu jsou identifikovány a studovány pro určení možností vážné nehody. Řídící zaměstnanci mohou použít tuto informaci ke kontrole potenciálního rizika a k zabránění případné nehody. Blokový diagram procesu FMEA je zobrazen na Obr. 13.



Obr. 13: Blokový diagram procesu FMEA

Použití FMEA

- Vývoj požadavků systému, které minimalizují pravděpodobnost selhání
- Vývoj metod návrhu a testování systému k zajištění, že selhání byli odstraněny.
- Zhodnocení požadavků zákazníka k zajištění, že nezaviní zvýšení potenciálních selhání.
- Identifikace určitých charakteristik návrhů, které přispívají k selhání a minimalizace nebo eliminace těchto efektů.
- Vystopování a ovládnutí potenciálních rizik v návrhu. To pomáhá předejít stejným chybám v budoucích projektech.
- Zajištění, že jakákoliv chyba, která nastane, nezraní uživatele nebo nepoškodí systém

Výhody

- Zlepšení kvality, spolehlivosti a bezpečnosti produktu
- Zlepšení prestiže a konkurenceschopnosti společnosti
- Zvýšení spokojenosti uživatele
- Snížení doby a ceny vývoje systému
- Shromáždění informací ke snížení budoucích selhání, získání znalostí o sytému
- Snížení potenciálních záručních oprav
- Včasná identifikace a eliminace potenciálních způsobů selhání
- Důrazná prevence problémů
- Minimalizace pozdních změn a nákladů s tím spojených

- Prostředek pro týmovou práci a výměnu nápadů mezi pracovníky

Nevýhody

V některých případech může FMEA identifikovat pouze hlavní způsoby selhání v systému. Pro některé druhy analýz je mnohem vhodnější FTA.

Druhy FMEA

- Proces: analýza výrobního a kompletačního procesu
- Design: analýza produktu před spuštěním výroby
- Koncept: analýza systémů nebo subsystémů v časných stádiích návrhu
- Vybavení: analýza strojů a designu vybavení před pořízením
- Služby: analýza procesů a služeb systému, před tím než je poskytnut zákazníkovi
- Systém: analýza funkcí celého systému
- Software: analýza funkcí softwaru

5.4 Hazard and operability studies (HAZOP)

HAZOP technologie byla vyvinuta v ICI v šedesátých letech jako metoda analýzy rizik v chemických továrnách a jejich velínech. Později byla CIA (Chemical Industries Association) rozvinuta a publikována, a teď je široce používána v množství aplikací včetně těch založených na použití počítačů (Kletz, 1995)[2].

Typicky je analýza vedena týmem od čtyř do osmi inženýrů, včetně expertů jak v aplikační oblasti tak v přímém vývoji systému. Vedoucí osoba tohoto týmu obyčejně bývá inženýr s rozsáhlým zaškolením v použití tohoto a dalších technik analýzy rizik. Tým začíná od základních specifikovaných operací systému a zkoumá vliv odchylek od těchto normálních operací. Pro každou odchylku hledá tým odpovědi na sérii otázek a zjišťuje, zda může daná odchylka nastat, a pokud ano, zda může vést ke vzniku rizika. Pokud je možný hazard detekován, dojde k dalšímu testování pro zjištění, kdy to může nastat, co s tím může být uděláno a kdy musí dojít k nápravným opatřením.

Analýza HAZOP začíná zjišťováním spojení mezi součástkami v systému a určováním odpovídajících vlivů. Za tyto vlivy může být považován fyzický tok materiálu od jedné komponenty ke druhé jako v případě chemické továrny, nebo mohou představovat tok elektřiny, signálů nebo dat. Takové toky jsou uváděny jako „entity“. Každá entita obsahuje určitý počet atributů, které určují správnost systémových operací. Například výměna dat mezi dvěma komponentami může mít atributy vztažené k hodnotě vyměněných dat a k bitovému toku jejich komunikace. Odchylky od hodnot těchto atributů v designu systému mohou mít důsledky na správný běh systému.

Analýza je založena na pečlivém a systematickém zkoumání možných odchylek od každého z identifikovaných atributů. Ve snaze strukturalizovat proces vymezení chyby je používána skupina „pomocných slov“ pro vyjádření konkrétních typů odchylek. Byl vytvořen obecně použitelný seznam pomocných slov. Těmi jsou:

ne
více
méně
stejně jako
část
opačně
jiný než

Zkušenosti ukázaly, že tento seznam je použitelný ve více druzích systémů, i v systémech založených na použití počítačů. Nicméně je nutné přidat další pomocná slova pro pokrytí vlivu času. Běžně se přidávají tyto slova:

dříve
později
před
po

Pro různá pomocná slova je použito různých interpretací v závislosti na druhu průmyslu a na oblasti použití. Z tohoto důvodu musí být význam, nebo významy každého z pomocných slov definovány jako část analýzy. Tab. 3 ukazuje, že pomocná slova jsou často interpretovány rozdílně mezi různými aplikacemi, a Tab. 4 ukazuje škálu možných významů dvou konkrétních pomocných slov, když jsou aplikovány na škálu atributů.

Pokud jsou jednouše aplikovány, každý atribut z každé interakce v systému může být zkoumán určením efektu každého relevantního pomocného slova. V praxi, zkušenosti jsou použity ke správnému výběru otázek pro každou z oblastí.

Pomocné slovo	Chemická továrna	Počítačový systém
Ne	Žádné části ze zamýšleného výsledku nebylo dosaženo	Nedojde k výměně dat nebo kontrolního signálu
Více	Značný nárůst fyzického množství	Velikost signálu nebo rychlost přenosu dat je moc vysoká
Méně	Značný pokles fyzického množství	Velikost signálu nebo rychlost přenosu dat je příliš nízká
Stejně jako	Zamýšlená aktivita nastane, ale s vedlejšími výsledky	Nadbytečná data jsou poslány spolu se zamýšlenou hodnotou
Část	Jenom část zamýšlené aktivity nastane	Nekompletní data jsou přenesena
Opačně	Nastane opak toho, co bylo zamýšleno, například zpětný tok v trubce	Polarita průběhu signálu se obrátí
Jiný než	Nic ze zamýšlené aktivity nenastane, a místo toho se stane něco jiného	Data jsou kompletní, ale nesprávná
Dříve	Nepoužito	Signál dorazí příliš brzo vzhledem k hodinovému taktu
Později	Nepoužito	Signál dorazí příliš pozdě vzhledem k hodinovému taktu
Před	Nepoužito	Signál dorazí dříve než se předpokládá v dané sekvenci
Po	Nepoužito	Signál dorazí později než se přepokládá v dané sekvenci

Tab. 3: Možná interpretace pomocných slov v různých aplikacích

Atribut	Pomocné slovo	Možný význam
Tok dat	Více	Více dat projde než se očekává
	Méně	Méně dat projde než se očekává
Rychlost přenosu dat	Více	Rychlost přenosu dat je příliš vysoká
	Méně	Rychlost přenosu dat je příliš nízká
Hodnota dat	Více	Hodnota dat je příliš vysoká
	Méně	Hodnota dat je příliš nízká
Doba opakování	Více	Doba mezi obnovením výstupu je příliš vysoká
	Méně	Doba mezi obnovením výstupu je příliš nízká
Doba odezvy	Více	Doba odezvy je větší než je požadováno
	Méně	Doba odezvy je menší než je požadováno

Tab. 4: Možná interpretace pomocných slov pro škálu atributů

Výsledky analýzy jsou zaznamenány v tabulce, který zobrazuje zjištění a doporučení výzkumného týmu.

Finálním stupněm procesu HAZOP je určit prioritu výsledků pro identifikaci oblastí které vyžadují podrobnější zkoumání. Jakmile je identifikováno riziko, může být vhodné použití stromu chyb pro prezentaci získaných dat.

HAZOP technologie může být také aplikována v oblasti software (Chudleigh and Catmur, 1992)[2]. V tomto případě mohou vhodné atributy zahrnout „hodnota dat“, „hodnota ukazatele“, algoritmus“ a „časování“, a vhodná pomocná slova mohou být obohacena o „nesprávný“, „příliš rychle“ a „příliš pomalu“. V počítačovém systému se předpokládá, že analýza je zpočátku řízena jako instalace celku. To je později následováno více detailní studií součástek počítače a jeho softwaru.

Je nezbytné být důkladný při zjišťování příčin těchto odchylek. Odchylka je považována za reálnou pokud jsou zde dostačující důvody věřit že tato odchylka nastane. Nicméně by měly být uvažovány pouze věrohodné příčiny. Záleží na posouzení celého týmu zda lze zahrnout i událost s velmi malou pravděpodobností vyskytnutí se. A proto musí tým mít dobrý úsudek v rozhodování, která událost má nízkou pravděpodobnost vyskytnutí se, takže věrohodné příčiny nejsou přehlédnuty.

Existují tři základní typy příčin chyby:

1. Lidská chyba, což je výsledek zanedbání nebo úmyslného poškození systému operátorem, designérem, konstruktérem nebo jinou osobou, a tím vytvořením rizika, které může vyústit například v únik nebezpečného nebo hořlavého materiálu.
2. Chyba zařízení, kdy mechanické, strukturální nebo operační selhání může například vyústit v únik nebezpečného nebo hořlavého materiálu.
3. Vnější události, při kterých předměty operující v okolí zařízení ovlivňují činnost jednotky a zvyšují riziko selhání. Vnější události zahrnují rušení přilehlými jednotkami ovlivňující bezpečný provoz jednotky, ztráta vybavení a vystavení počasí a seismické aktivitě.

Primárním účelem HAZOP je vytvoření scénářů, který vedou k vzniku nebezpečné události. Při vytváření těchto scénářů je vždy důležité určit co nejpřesněji všechny následky

jakýchkoliv možných věrohodných příčin. To poslouží dvěma záměrům. Za prvé pomůže určit klasifikaci rizika v HAZOP analýzách, které odhalili několikanásobné riziko, takže může být určena priorita při zkoumání jednotlivých rizik. A za druhé pomůže určit, zda konkrétní odchylky mohou způsobit problémy při provozu nebo možné riziko.

Pokud tým dojde k závěru, že následkem konkrétní odchylky budou pouze provozní problémy, pak by zde mělo zkoumání skončit a tým by se měl zaměřit na další příčinu, odchylku nebo část systému. Pokud tým zjistí, že odchylka způsobí nebezpečnou situaci, pak by měly být vytvořeny bezpečnostní opatření.

Bezpečnostní opatření by měly zahrnuty pokaždé, když tým určí, že kombinace příčin a následků představuje věrohodné riziko provozu. Co se považuje za bezpečnostní opatření může být shrnuto následujícími kritérii:

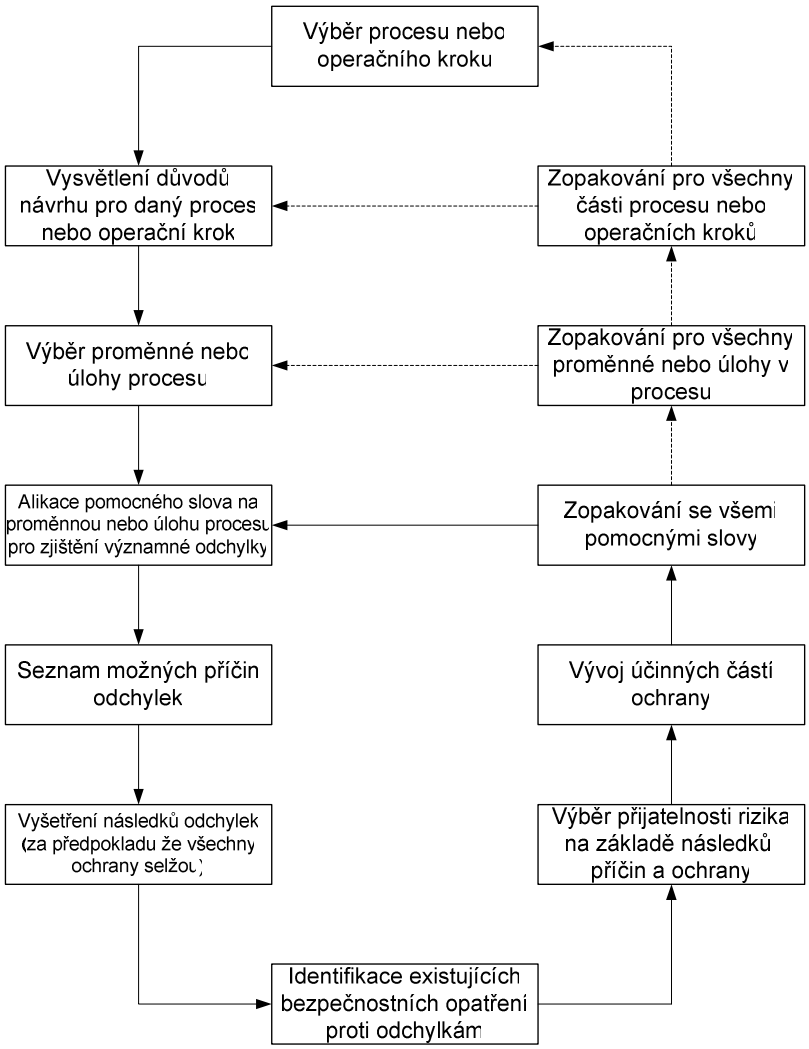
1. Systémy, inženýrské designy a procedury, navržené pro zabránění vzniku rizikové situace.

2. Systémy, které jsou navrženy pro detekci a včasné varování na počáteční podmínky vedoucí k rizikové situaci.

3. Systémy nebo procedury, které zmírňují následky vzniku rizikové situace.

Tým by měl být pečlivý při výběru bezpečnostních opatření. Analýza rizika vyžaduje ohodnocení následků selhání provozních a administrativních kontrol, takže musí být provedeno pečlivé zhodnocení toho, zda tyto kontroly mohou nebo nemohou být považovány za bezpečnostní opatření. Navíc by tým měl zvážit reálné opakující se selhání a náhodné události při rozhodování, zda tyto bezpečnostní kontroly budou skutečně fungovat v případě výskytu selhání.

Výsledkem analýzy HAZOP jsou doporučení týmu provádějícího analýzu, která zahrnují identifikaci rizika a doporučení změn v návrhu, procesech apod. pro zvýšení bezpečnosti systému. Odchylky během běhu, startu, vypnutí a údržby systému jsou prozkoumány týmem a zahrnuty v HAZOP zprávě. Na následujícím diagramu je zobrazen blokový diagram procesu HAZOP.



Obr. 14: Blokový diagram procesu HAZOP

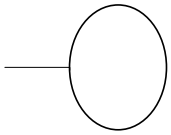
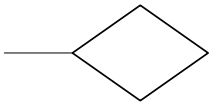
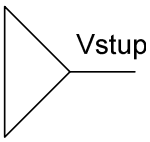
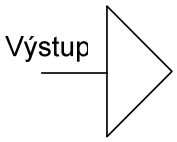
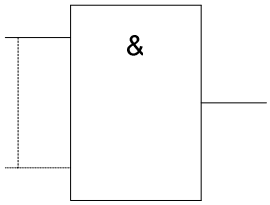
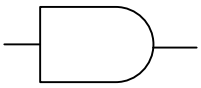
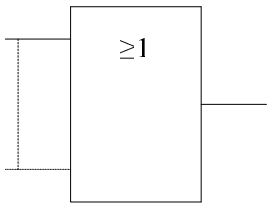
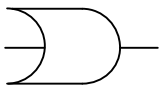
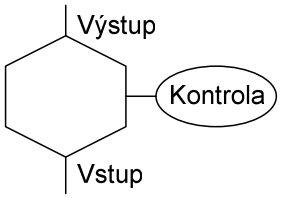
5.5 Fault Tree Analysys (FTA)

Analýza pomocí stromu chyb používá symboly definované v normě IEC 1025 [2] viz Tab. 5. Strom je konvenčně nakreslen s hlavní událostí na pravé straně diagramu. Jednoduchá ukázka chybového stromu dle této normy je zobrazena na Obr. 15. Diagram zobrazuje podmínky selhání topného systému, který používá buď kapalné nebo pevné palivo. Tento systém potřebuje elektrické napájení kontrolního systému a větrání. Je zřejmé, že systém selže, pokud selže elektrické napájení nebo pokud dojdou oba zdroje paliva.

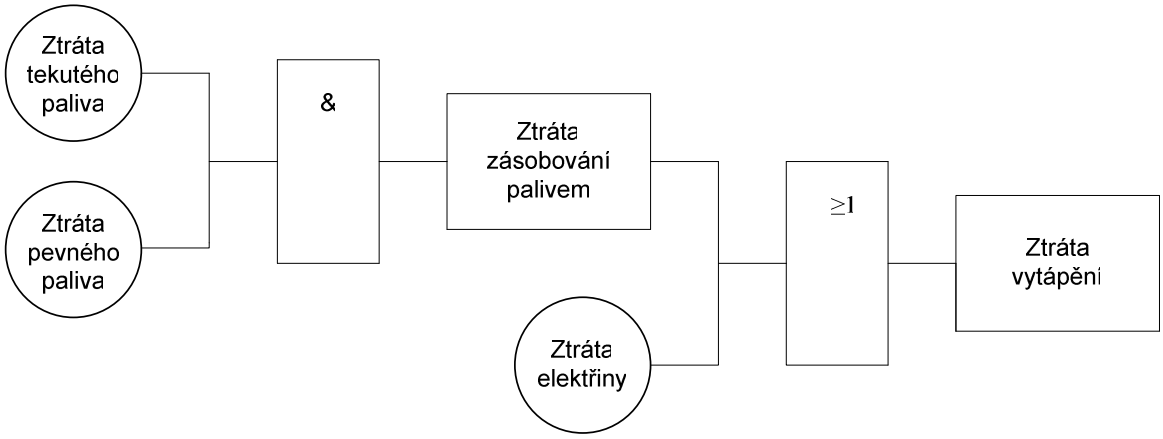
Použitím alternativních symbolů z IEC 1025 je hlavní událost většinou nakreslena na vrcholu diagramu. Při použití této formy zápisu jsou symboly velice podobné symbolům používaným v diagramech elektrické logiky, což strom událostí činí snáze pochopitelný pro každého kdo je obeznámen s elektrickými obvody. Obr. 16 zobrazuje strom chyb z Obr. 15 zakreslen tímto způsobem.

Při konstrukci stromu chyb je vhodné rozlišovat chyby různých tříd. Chyby mohou být klasifikovány jako primární, sekundární nebo příkazové chyby (Veselý, 1981)[2]. Primární chyby vznikají když součástka selže během provozu v prostředí a za podmínek, pro které byla navržena. Tudíž když rezistor stanovený pro 1 W selže při průchodu proudu vytvářející menší výkon než je tato hodnota, byl zasažen primární chybou. Sekundární chyba nastane když

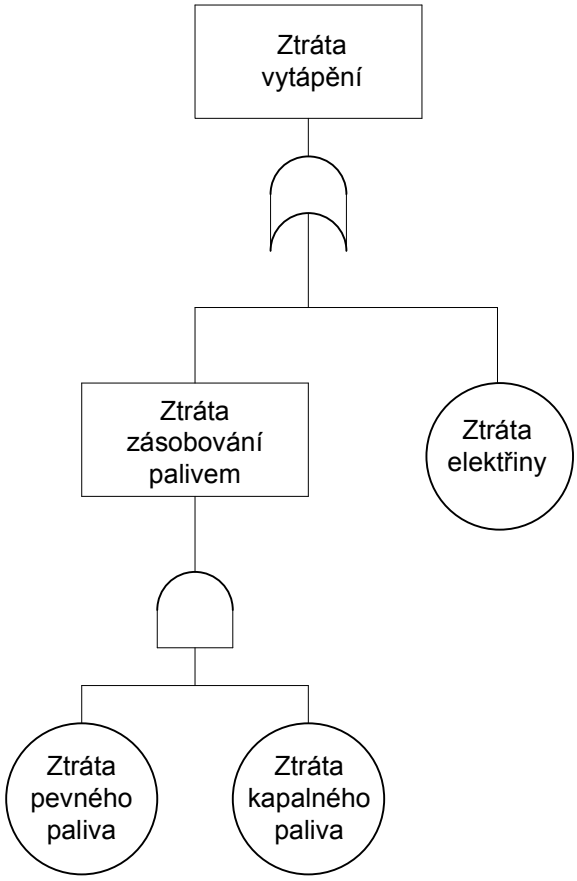
součástka selže za podmínek, které jsou mimo její navrženou pracovní oblast. Pokud náš 1 W rezistor selže při proudu vytvářející 10 W, dojde k sekundární chybě. Jak napovídá pojmenování, sekundární chyba je způsobena okolními vlivy uvažované součástky. Příkazová chyba se vyskytne, pokud součástka pracuje správně, ale za nesprávných okolností. Například pokud elektromagnet správně reaguje na nesprávný kontrolní signál, což je příkazová chyba, je důsledek činnosti způsoben zdrojem nesprávného signálu.

Preferovaný symbol	Alternativní symbol	Význam
		Chybová událost vychází z jiných událostí
		Základní událost, brána jako vstup
		Chybová událost, u které není plně znám její zdroj. Je brána jako vstup, ale její příčiny mohou být neznámé
		Trojúhelníkové symboly jsou používány ke spojení stromů. Symbol ‚vstup' znázorňuje vstup z jiného stromu.
		Symbol ‚výstup' se objevuje v místě hlavní události a znázorňuje, že v tomto místě se vytváří vstup do jiného stromu
		Výstupní událost nastane, pokud VŠECHNY vstupní události nastanou.
		Výstupní událost nastane, pokud NĚJAKÁ ze vstupních událostí nastane, ať už sama nebo v kombinaci s jinou.
		Kontrolní podmínky určují zda se vstupní hodnota objeví na výstupu.

Tab. 5: Symboly chybového stromu podle IEC 1025



Obr: 15: Jednoduchý strom chyb



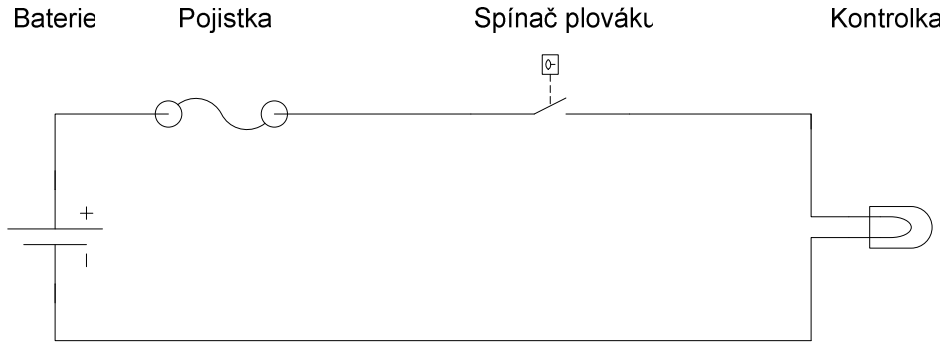
Obr.16: Strom chyb z Obr.15 nakreslený použitím alternativních symbolů podle IEC 1025

Ukážeme si dva praktické příklady konstrukce stromu chyb.

Příklad 1:

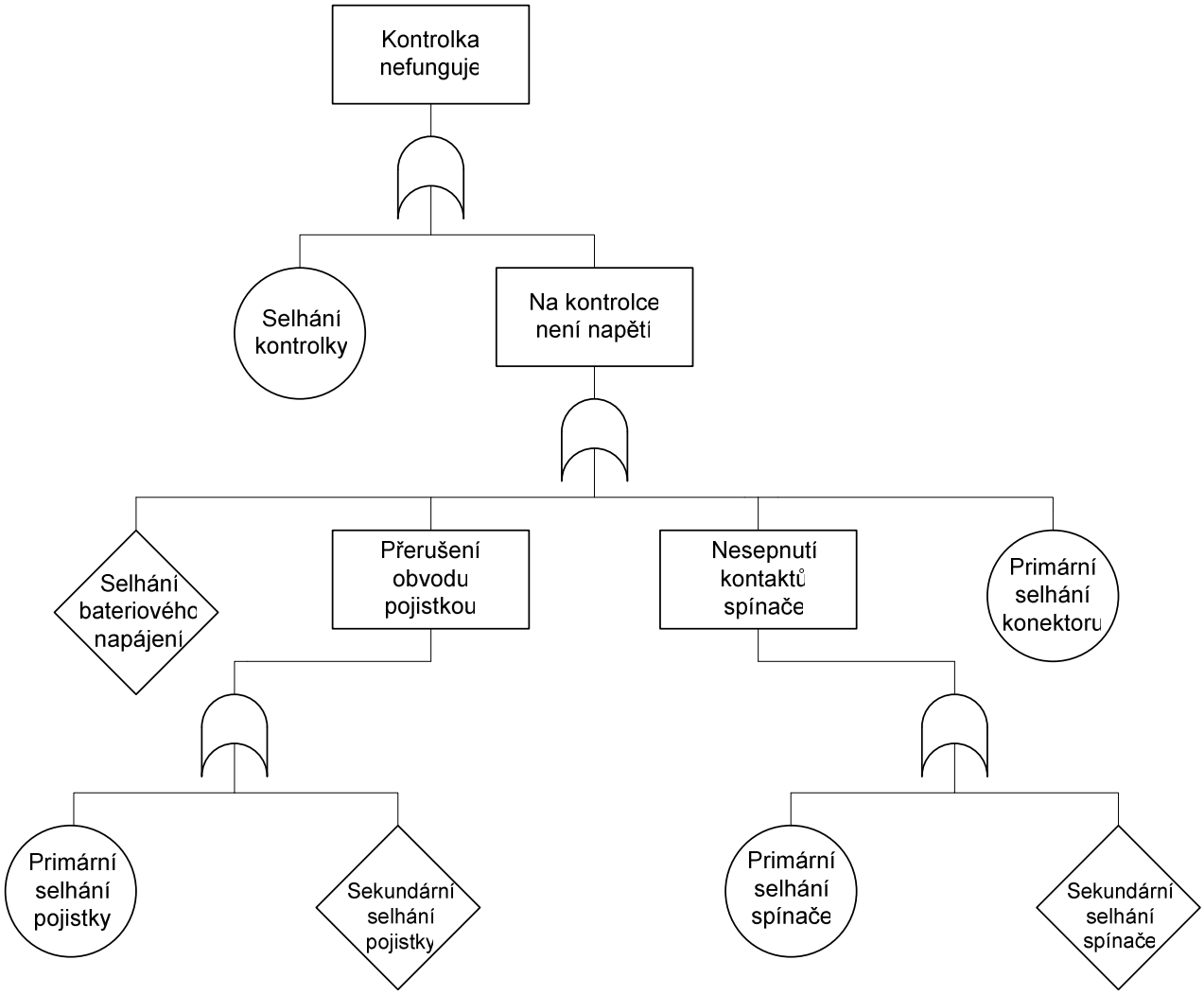
Konstrukce stromu chyb pro světelný varovný systém kontroly hladiny brzdné kapaliny v automobilu. Hlavní událostí je jednoduše kontrolka, která nesvítí když je hladina brzdové kapaliny nízká.

Na následujícím obrázku je návrh obvodu takového systému. Zobrazuje sériové zapojení baterie, pojistky, spínače plováku použitého k detekci hladiny brzdové kapaliny, a kontrolku. Obvod také obsahuje dráty a konektory.



Obr. 17: Návrh varovného systému kontroly hladiny brzdové kapaliny

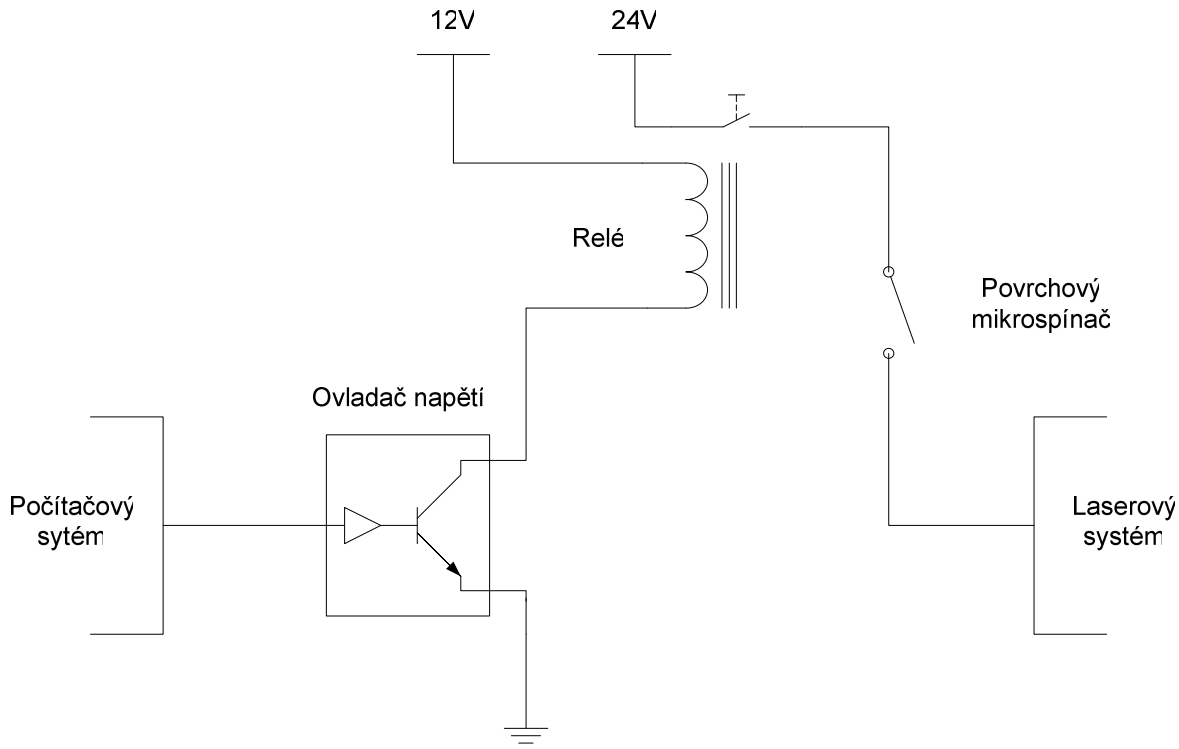
Možné ztvárnění stromu chyb je zobrazeno na Obr. 18. Strom ukazuje, že kontrolka nebude svítit, když nastane jedna ze dvou podmínek. První podmínkou je selhání kontrolky, jako třeba prasklé vlákno. Druhou podmínkou je, že z nějakého důvodu není na kontrolce žádné napětí. Tato druhá podmínka je dále analyzována, a jsou identifikovány 4 možné příčiny. Těmi jsou: selhání napájení z baterie, přerušení obvodu spálením pojistky, selhání spínače nebo selhání konektoru primárního kabelu. Selhání bateriového napájení může být způsobeno mnoha příčinami, a strom to ponechává jako událost, která není vysledována až ke zdroji. Selhání pojistky může mít buď primární nebo sekundární příčinu. Zde primární příčina představuje chybu uvnitř pojistky, způsobující její shoření za normálních provozních podmínek a sekundární příčina představuje selhání pojistky způsobené nadměrným proudem. Příčina nadměrného proudu je ponechána jako událost která není vysledována ke zdroji. Selhání spínače může být způsobeno primární chybou samotného spínače, nebo jinými druhotnými efekty, jako třeba přítomnost cizího objektu v nádrži, zabraňujícím sepnutí spínače. Stejně jako u sekundárního selhání pojistky, i zde není událost vysledována ke zdroji.



Obr. 18: Strom chyb pro varovný systém kontroly hladiny brzdné kapaliny

Příklad 2:

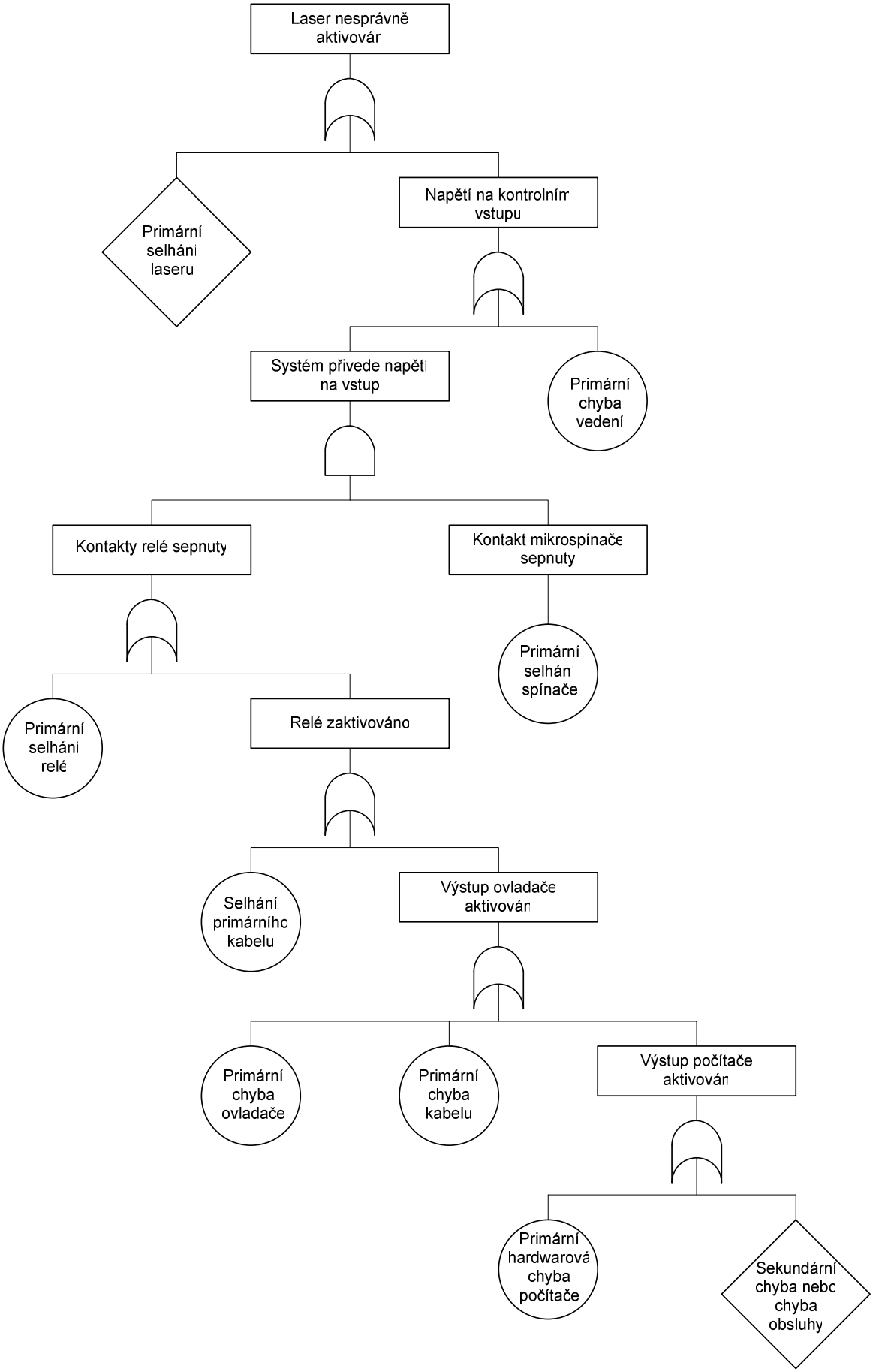
Konstrukce stromu chyb pro laserový vypalující systém z Obr. 19. Laser je ovládán z počítače přes ovladač napětí a elektromagnetické relé. Pálení laserem je spuštěno lidskou obsluhou, která se nejprve ujistí, zda je pálicí oblast v pořádku a zda je bezpečnostní kryt uzavřen. Mikrospínač připojený ke krytu odpojí ovládací signál laseru pokud se kryt otevře. Hlavní událostí je aktivace laseru když je bezpečnostní kryt otevřen.



Obr. 19: Ovládání laserového kontrolního systému

Možný strom událostí pro tento systém je zachycen na Obr. 20. Spuštění laseru když je bezpečnostní kryt otevřen může nastat jako důsledek primárního selhání laseru, a pak laserový systém může fungovat i bez kontrolního signálu. Tato událost zde není dále vyšetřována, a je zaznamenána jako událost bez nalezeného zdroje. Druhou možností je, že laser může být aktivován protože byl na jeho vstup přiveden kontrolní signál. Možnými příčinami druhé události je zapůsobením nějakého druhu chyby se na kabeláž přivedlo napětí na vstup, nebo že jiné části systému přivedli tento signál. Tato podmínka nastane pouze pokud jsou kontakty relé a mikrospínače sepnuty. Pokud uvažujeme pouze situaci, kdy je laser spuštěn při otevřeném krytu, pak budou kontakty mikrospínače sepnuty pouze v případě primární chyby mikrospínače. Podmínky pro zavření kontaktů relé jsou složitější. K tomu může dojít pouze díky primární chybě relé nebo protože cívka relé byla aktivována. Aktivace cívky relé může být způsobena aktivací ovladače napětí, ale také může být způsobena chybou. Ovladač napětí může být aktivován jako důsledek primární chyby samotného ovladače, chyby na kabelu nebo kontrolním signálem z počítače. Protože by počítač neměl spustit laser pokud je kryt otevřen, tak pokud k tomu dojde, je to způsobeno uvnitř počítače primární chybou, sekundární chybou nebo chybou obsluhy. To může být nějaký druh hardwarové nebo softwarové chyby v procesoru, selhání některého externího hardwaru nebo systému, nebo chyba lidské obsluhy. To je opět ponecháno jako událost bez zjištění příčiny.

Je dobré si všimnout, že chyby kabeláže, které mohou zahrnovat chyby konektorů nebo spojení, vytvářejí velký počet větví na stromu chyb. V závislosti na umístění těchto chyb to může mít velmi rozdílné efekty na provoz systému. Je také zajímavé zmínit, že přítomnost mikrospínače na bezpečnostním krytu vytváří silnou ochranu systému proti chybě počítače nebo obsluhy. Navíc chrání sytém před mnoha druhy hardwarového selhání. Nicméně nechrání systém před následky selhání samotného laseru, před selháním mikrospínače a před určitými chybami kabeláže. Žádná ochrana není perfektní.



Obr. 20: Strom chyb laserového kontrolního systému

5.6 Analýza rizika během vývoje systému

Z toho, co jsme probrali v této a předchozích kapitolách je jasné, že analýza rizika hraje důležitou roli při vývoji jakéhokoliv „embedded“ počítačového systému. Její výsledky neovlivňují pouze design systému, ale také použité vývojové metody. Je proto jasné že analýza rizika musí být provedena v ranné fázi vývoje, protože její výsledky mají veliký vliv na všechny aspekty projektu. Nicméně by bylo chybou považovat analýzu rizika za proces provedený pouze na začátku projektu. Ve skutečnosti se analýza zaobírá ne jenom charakteristikou systému, ale také detaily v designu systému. A proto pokud předběžná analýza ukáže že je systém bezpečný, analýza rizika normálně probíhá dále skrze celý vývojový proces. Podstatou této práce a v ní zahrnutého úsilí je určení několika faktorů, včetně stupně rizika spojeného s provozem systému.

Pokračující vyhledávání a analýza rizika se sestává z množství fází, které nastávají během různých úrovní vývojového procesu. Těmto fázím jsou dávána různá jména, my použijeme terminologii převzatou od (MoD, 1995a)[2]. Hlavní fáze analýzy rizika pro typický projekt vývoje systému s důrazem na bezpečnost je zobrazen na Obr. 21.

Předběžná identifikace rizika PHI (Preliminary hazard identification)

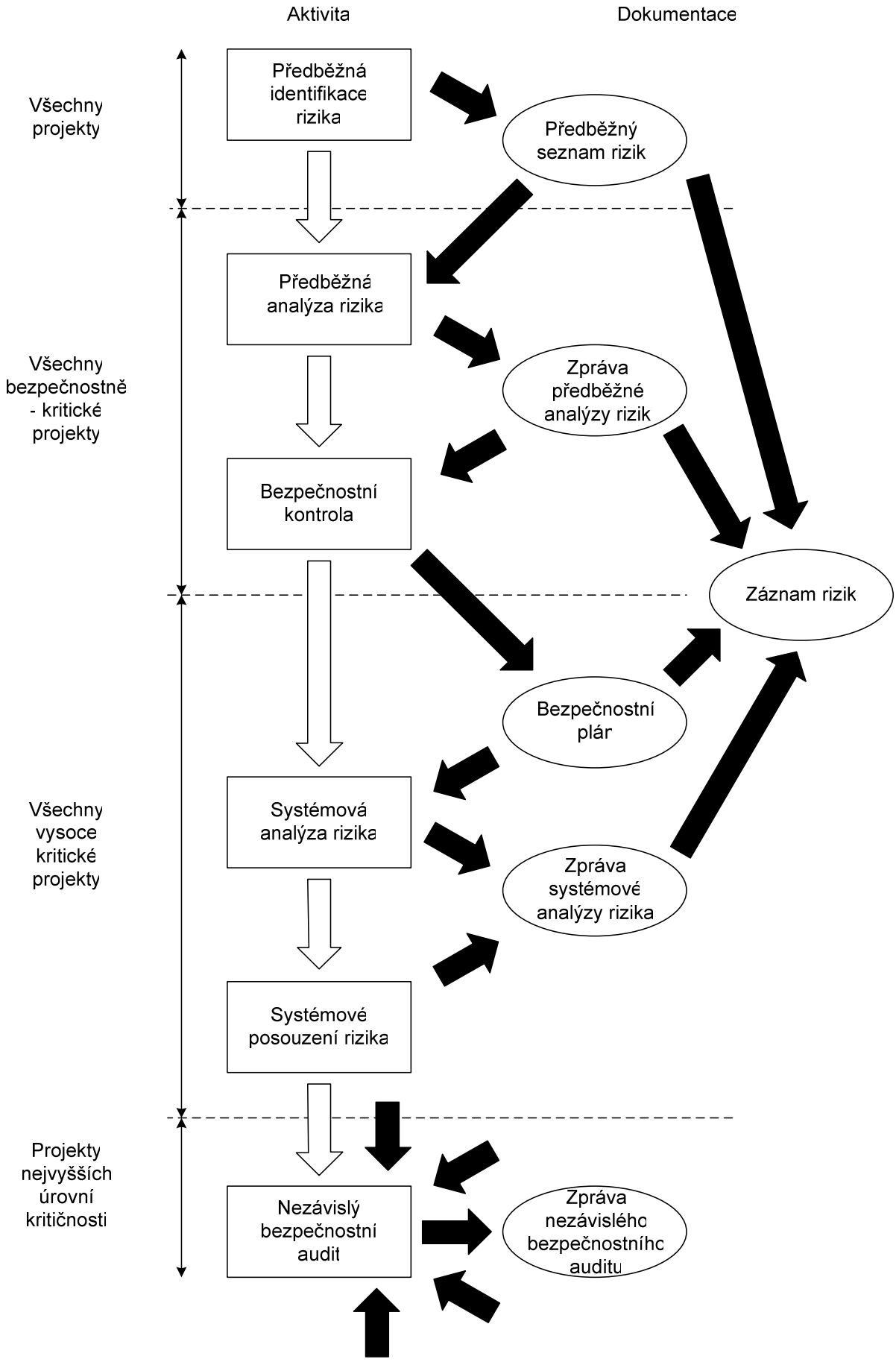
Vliv značného rizika má důsledky pro design, vývoj a použití jakéhokoliv systému. Je proto nezbytné identifikovat riziko již ve velmi časné fázi vývoje systému, aby bylo možné podniknout nezbytné kroky. Tyto kroky mohou zahrnovat změnu podstaty systému a tím odstranit důsledky rizika. Důležitost tohoto procesu naznačuje, že by všechny embedded systémy měli být vystaveny procesu předběžné identifikace rizika k určení všech postižených oblastí. Měl by být proveden v časné fázi vývoje systému – obvykle během tvorby koncepce projektu. Systematické studium operačních i chybových podmínek by mělo být provedeno za použití osvědčených metod, jako například HAZOP. V aplikacích, kde jsou podobné systémy už v provozu, mohou být data z předchozích nehod nebo událostí použitelné jako důležité vstupní údaje procesu identifikace rizika.

Výsledky PHI jsou zaznamenány v předběžném listu rizik. V aplikacích, které jsou shledány být náchylnými k chybovosti je tento dokument použit jako vstupní údaj při pozdějších stádiích analýzy rizika. Pokud zkoumání nenalezne žádné významné riziko, pak není další analýza vyžadována a systém je označen jako relativně bezpečný. V tomto případě by měl být záznam předběžné analýzy rizika zachován jako důkaz provedeného procesu.

Předběžná analýza rizika PHA (Preliminary hazard analysys)

Předběžná analýza rizika pracuje s rizikem identifikovaným v PHI fázi a tyto údaje podstoupí detailnějšímu studiu pomocí HAZOP nebo jiné systematické techniky. Každé riziko je uvažováno v souvislosti s funkčními požadavky systému pro identifikaci bezpečnostních důsledků a zhodnocení alternativ v návrhu systému. Každé zjištěné riziko je zaznamenáno v **záznamu rizik**, kterýžto reprezentuje výsledný záznam bezpečnostních problémů systému.

Kromě identifikace rizik se tato analýza také snaží, v rozsahu jakém je to možné v tomto ranném stádiu, klasifikovat kritičnost rizika a přiřazení stupně integrity požadované každou z hlavních funkcí.



Obr. 21: Typický průběh analýzy rizik při vývoji bezpečnostně kritického systému

Údaje zjištěné během PHA jsou zdokumentovány ve **zprávě předběžné analýzy rizik.** Ta poskytuje velký zdroj informací o systému a jeho možných rizikách, a to včetně následujícího:

- stručný popis systému a jeho pracovního prostředí
- celkový přehled funkcí systému a bezpečnostních vlastností
- bezpečnostní cíle systému
- ospravedlnění rizik a přiřazených úrovní integrity
- cílové míry chybovosti a bezpečnostních hladin
- zdroje všech dat použitých v analýze
- seznam použité literatury

Předběžná analýza rizika je použita jako vstupní data pro fázi tvorby bezpečnostních požadavků a jako příručka při prvních návrzích architektury systému a technik. PHA také provází pozdní analýzy rizik.

Bezpečnostní kontroly

Během procesu vývoje je projekt vystaven množství kontrol, které pohlížejí na všechny aspekty bezpečnosti. Během těchto kontrol jsou brána data z různých reportů a analyzována. Výsledky jsou zaznamenány v záznamu rizik pro budoucí použití. První z těchto bezpečnostních kontrol se často provádí hned fázi PHA. Bezpečnostní kontroly jsou často prováděny jako součást kontrol projektového designu v pozdějších stádiích vývoje.

Jeden z úkolů provedených v předběžné analýze rizika je ohodnocení integrity požadavků na systém. Pokud tato analýza a s ní spojené kontroly bezpečnosti ukáže, že systém vykazuje relativně nízkou míru kritičnosti, je pravděpodobné, že další více detailní analýza nebude nutná nebo ekonomicky ospravedlnitelná. V tomto případě bude vývoj pokračovat použitím metod vhodných přiřazenému stupni integrity.

Bezpečnostní plán

Pro všechny systémy které se ukázaly zasaženy středním nebo vysokým stupněm kritičnosti, je vytvořen bezpečnostní plán určující způsob dosažení bezpečnosti a definující strukturu řízení zodpovědnou za další analýzu rizika. Bezpečnostní plán může také jmenovitě určovat klíčové zaměstnance. To je zvláště výhodné při přiřazování zodpovědností v projektu prováděném množstvím společností nebo konsorcií.

Bezpečnostní plán je použit k dokumentaci detailního plánování bezpečnosti projektu a použitých kontrolních měřítek. Také bude zahrnovat záznam různých standardů a následných praktik s detailním popisem způsobu dosažení požadavků tohoto dokumentu. Plán je upravován během projektu.

Systémová analýza rizika

Systémy, které vyžadují vysokou úroveň integrity budou vystaveny dalšímu zkoumání ve formě systémové analýzy rizika. Cílem této práce je rozšíření a předefinování nálezů předběžné analýzy rizika uvážením detailních funkcí systému a součástek které je implementují. V počátku analýza pracuje s požadavky systému a s daty z PHA. Jak se projekt vyvíjí, analýza se začne více zajímat součástkami a částmi systému, které byli navrženy pro splnění těchto požadavků.

Ve většině případů je požadován výběr technik analýzy rizika pro úplné pokrytí možných rizik.

Nezávislý bezpečnostní audit

V projektech zabývajících se systémy s nejvyššími úrovněmi kritičnosti je zaměstnán nezávislý tým expertů provádějících bezpečnostní audit. Tento tým si bere data ze záznamu rizika a z různých reportů analýzy rizik a nezávisle prověřuje důkladnost práce a správnost výsledných řešení. Výsledek auditu je zdokumentován ve zprávě nezávislého bezpečnostního auditu, který vytvoří podstatnou část dokumentů potřebných k dosažení certifikace.

Použití nezávislého názoru v hodnocení je také doporučeno při vývoji systému s více mírnými úrovněmi kritičnosti. V takovýchto případech je užitečné identifikovat různé stupně nezávislosti mezi osobou provádějící posudek a osobou provádějící vývoj. Tyto osoby mohou být klasifikovány jako:

1) nezávislá osoba
2) nezávislé oddělení
3) nezávislá organizace

Zvýšené úrovně nezávislosti pomáhají zvýšit naše přesvědčení ve správnost systému. Stupeň nezávislosti bude nicméně často omezen z ekonomických důvodů. Tato oblast podléhá různým mezinárodním standardům, jejichž popsání je mimo rozsah této práce.

Použití nezávislého týmu posuzovatelů klade velký důraz na kompletnost dokumentace projektu a vyžaduje velkou míru spolupráce od všech zainteresovaných pracovníků. Všechny výsledky a analýzy zaznamenané v různých dokumentech musí být možné vysledovat zpět ke zdroji a tím umožnit nezávislou verifikaci. Toto ustanovení je nazýváno **stopa auditu**.

6 Techniky testování procesorů

Spolu se zvyšujícím se kmitočtem moderních procesorů rostou i nároky na kontrolu bezchybné funkcionality procesorů a tím i na způsoby testování procesorů. Existuje mnoho způsobů testování procesorů, v současné době se uplatňuje především testování za běhu, kdy je možné daný procesor testovat i během provozu. S dosažením pracovní frekvence v řádu GHz se testování procesorů za běhu stalo velice náročné na vnější testery. Jedním z řešení je **built-in self-test**, který snižuje nároky na vnější testery. Ovšem z důvodů závislosti těchto testů na náhodných testovacích sekvencích, nejsou současné technologie BIST schopné vypořádat se s rozsáhle designovanými obvody bez značných režijních nároků. Podíváme se podrobněji na některé techniky:

6.1 Built-in self-test (BIST)

Mechanismus bulit-in self-test v integrovaném obvodu je funkce, která prověřuje všechny nebo část funkcí daného obvodu. Mechanismus BIST se například používá ve vylepšeném sběrnicovém systému pro ověření funkcionality. Na vyšší úrovni je možné BIST srovnat s funkcí Power-On Self-Test (POST) systému BIOS v osobních počítačích, který provádí self-test pamětí RAM a sběrnic při startu počítače.

Hlavním účelem BIST je snížení složitosti obvodu a tím i snížení ceny a závislosti na externím testovacím vybavení. BIST snižuje cenu dvěmi způsoby: snížením délky testovacího cyklu a snížením složitosti nastavení testu, a to tak, že sníží počet vstupních a výstupních

signálů, které musejí projít kontrolou testera. Oboje vede ke snížení hodinové sazby automatického testovacího systému.

Kromě výstupního testování může být BIST navržen k provádění diagnostiky jednotlivých zařízení nebo celých systémů. Například při startu většina moderních počítačových periferních zařízení (tiskárny, monitory) provádí omezenou diagnostiku sebe sama. Selhání této diagnostiky je hlášeno uživateli. Někdy tyto diagnostiky zapojí BIST v hardware k ověření funkčnosti sběrnic, I/O obvodů a hlavních registrů.

Název BIST a celý tento koncept vznikl s myšlenkou zahrnutí generátoru pseudonáhodných čísel a kontrolního součtu CRC do integrovaného obvodu. Pokud všechny registry fungující v integrovaném obvodu podléhají jednomu nebo více vnitřním scanům, potom funkce registrů a kombinatorické logiky mezi nimi vygeneruje unikátní CRC součet na dostatečně velikém vzorku náhodných vstupů. Takže jediné co integrovaný obvod musí udělat, je ukládat předpokládané CRC součty a otestovat je poté, kdy je vytvořen dostatečně veliký vzorek v generátoru pseudonáhodných čísel. Koncept vnitřního scanování je popsán ve standardu IEEE 1149.1[4].

Existuje několik specializovaných verzí BIST, které se liší svou funkcí nebo účelem použití:

- Programovatelný Built-In Self-Test (PBIST)
- Paměťový Built-In Self-Test (MBIST)
- Logický Built-In Self-Test (LBIST)
- Analogový a Mixed-Signálový Built-In Self-Test(AMBIST)

6.2 Automatic Test Pattern Generation (ATPG)

ATPG je metoda používaná k nalezení vstupní nebo testovací sekvence která, pokud je aplikovaná na digitální obvod, umožňuje testerům rozlišit mezi korektním chováním obvodu a chybným chováním, způsobeným defektem obvodu. Generované sekvence jsou použity k testování polovodičových jednotek při výrobě a v některých případech pomáhají při určování příčiny selhání. Efektivita ATPG je měřena množstvím modelovaných defektů, nebo chybových modelů, které jsou detekovány generovanými sekvencemi. Tyto měření obecně indikují kvalitu testu (vyšší s detekcí více chyb) a dobu aplikace testu (vyšší s více sekvencemi). Výkonnost ATPG je dalším důležitým faktorem. Je ovlivněna zvažovaným chybovým modelem, typem testovaného obvodu, úrovní abstrakce použité pro reprezentaci testovaného obvodu a požadovanou kvalitou testu.

Základy ATPG

Defekt je vlastně chyba při výrobě, která se promítne do činnosti obvodu. Chybový model je matematický popis toho, jak defekt ovlivní chování systému. K detekci chyby pomocí testovací sekvence dojde, pokud při aplikaci sekvence na obvod je na jednom nebo více primárních výstupech zachycena logická hodnota, která se liší od hodnoty požadované podle návrhu obvodu. ATPG proces pro zaměřené chyby se sestává ze dvou fází: aktivace chyby a propagace chyby. Aktivace chyby ustanoví hodnotu signálu na místě chybového modelu, která je opačná hodnotě vytvářené chybovým modelem. Propagace chyby posune výslednou hodnotu signálu, nebo chybového efektu, dopředu pomocí vyznačení cesty z místa chyby do primárního výstupu.

ATPG může selhat při hledání vhodného testu pro konkrétní chybu v nejméně dvou případech. Za prvé, chyba může být skutečně nedetekovatelná, pokud neexistují žádné

sekvence které mohou detekovat tuto konkrétní chybu. Klasickým příkladem je redundantní obvod, navržený tak, že jedna chyba nemůže způsobit změnu výstupu. V takovém obvodu je jakákoliv jediná chyba z podstaty nedetekovatelná. Za druhé, je možné že potřebná sekvence existuje, ale použitý algoritmus ji nemůže najít. Její vyhledání trvá tak dlouho, že ATPG prostě nahlásí, že tam chyba není.

Sekvenční ATPG

Sekvenční ATPG hledá sekvence vektorů pro detekci konkrétní chyby pomocí množiny všech možných vektorových sekvencí. Byli vytvořeny různé hledací strategie a heuristiky k nalezení kratší sekvence a nebo k rychlejšímu hledání. Nicméně podle zkušeností žádná strategie nebo heuristika nedokáže zastoupit jiné u všech aplikací nebo obvodů. Toto zjištění znamená, že generátor testů by měl zahrnout kompletní soubor heuristik.

Dokonce i jednoduchá trvající chyba vyžaduje sekvenci vektorů pro detekci v sekvenčním obvodu. Také díky přítomnosti paměťových elementů je ovladatelnost a rozpoznatelnost interních signálů v sekvenčním obvodu všeobecně mnohem složitější než v kombinatorických obvodech. Tyto faktory tvoří sekvenční ATPG mnohem složitějším než kombinatorické ATPG.

Díky veliké složitosti sekvenčního ATPG to zůstává náročným úkolem u velikých obvodů. Nicméně vykazují částečnou úroveň úspěchu a pro obvody, které jsou citlivé na rozsah velikosti nebo provozní režii, je řešením použití ATPG pro sekvenční obvody spolu s částečným scanováním obvodů.

Algoritmické metody

Testování velmi rozsáhlých integrovaných obvodů s vysokým pokrytím chyb je složitý úkol. Proto bylo vytvořeno mnoho rozdílných ATPG metod pro adresaci kombinatorických a sekvenčních obvodů.

- První generace testovacích algoritmů jako *booleovská rozdílnost* nebo *doslovný výrok* byli nepraktické pro implementaci v počítači
- **D Algoritmus** byl první algoritmus generování testů splňující paměťové požadavky. D Algoritmus zavedl D Notaci, která se dále používá ve většině ATPG algoritmů.
- **Path-Oriented Decision Making** (PODEM) je vylepšením D Algoritmu. PODEM byl vytvořen v roce 1981 kdy začaly být zřejmé nedostatky D Algoritmu díky změnám a rozvoji integrovaných obvodů. Které D Algoritmus nemohl obsáhnout.
- **Fan-Out Oriented** (FAN Algorithm) je vylepšením algoritmu PODEM. Omezuje hledací prostor ATPG a tím snižuje výpočetní čas a urychluje zpětné trasování.
- **Pseudonáhodné generování testů** je nejjednodušší metoda vytváření testů. Používá pseudonáhodný generátor čísel pro generování testovacích vektorů.

6.3 Deterministický funkční self-test

Výhodou této metody je to, že už z podstaty je deterministická. Zaměřením testu na zvládnutelné součásti s pomocí funkcionality procesoru má tato technologie schopnost pokrytí chyb jako deterministické testování a rychlost jako funkční testování. Původně byla odpovědnost při generování self-test programů nechána na test inženýrech. Později byla navržena technologie funkčního self-testování procesorů. Oba tyto postupy závisejí na generování a aplikaci sekvencí náhodných instrukcí na jádro procesoru. Jak využívají funkcionalitu procesoru, je pro dosažení vysokého pokrytí chyb třeba malý ztrátový režijní

výkon. Náhodné instrukce, ve srovnání s náhodnými sekvencemi, jsou mnohem efektivnější při testování procesorů, protože využívají spojení v procesoru, navržené při vývoji. Proto jsou už z podstaty deterministické. Tyto techniky nicméně vyžadují manuální úpravy self-test programů ve snaze o dosáhnutí maximálního pokrytí chyb. To znamená, že úspěšný self-test procesoru nemůže být z podstaty kompletně náhodný.

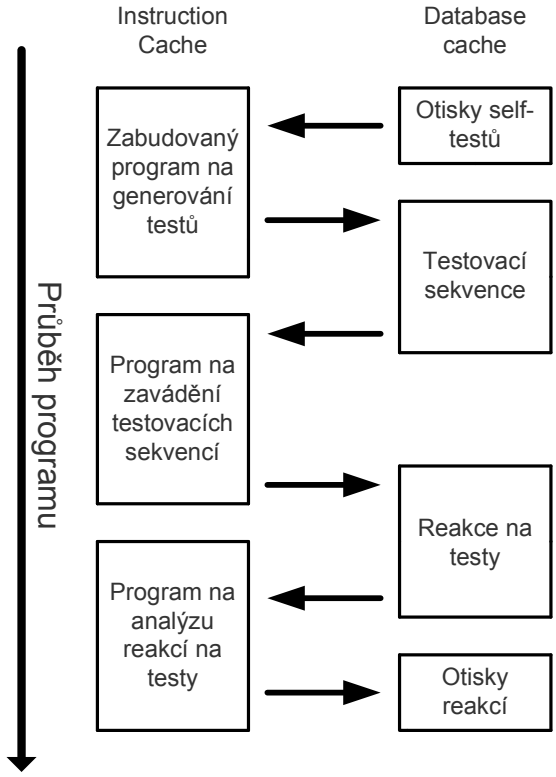
Tato metoda používá přístup rozděl a panuj. Počátečním krokem je generace realizovatelných testů jednotlivých součástek a shrnutí těchto testů do sekvence self-testu. Dalším krokem je self-test, což je testovací aplikace na chipu, která shromažďuje reakce na testovací vzorce za použití funkcionality testovacího procesoru. Na součástkové úrovni jsou testy zaměřeny na chyby struktury. Na procesorové úrovni se využívá funkcionalita procesoru pro aplikaci strukturových testů za běhu na každou ze součástek. Strukturové testy součástek jsou volány pomocí instrukcí procesoru. Proto musejí dodržet podmínky zavedené souborem instrukcí.

V počátečním kroku jsou vytvořeny testy pro jednotlivé součástky, jako je například ALU. Strukturové chyby jsou zaměřeny během generování testu součástek. Test může být deterministický i náhodný. Jestliže je zvolen soubor náhodných testů, charakterizuje se potřebná forma testu pro každou součástku jako otisk self-testu, což zahrnuje zdroj a nastavení generátoru pseudonáhodných čísel, a také množství generovaných testovacích sekvencí. Otisky self-testu jsou nahrány do paměti procesoru namísto skutečných testů před samotným testem. Pokud je zvolen soubor deterministických testů, jsou testy nahrány přímo do paměti procesoru před testem. Důvodem používání self-test podpisů je redukce doby nahrávání testovacích souborů a paměťových nároků při ukládání všech testovacích sekvencí ve stejnou dobu. Uložené testy jsou upřednostněny před pseudonáhodnými testy pouze pokud velikost testovacích souborů není o moc větší než velikost programu na generování pseudonáhodných čísel. Otisky self-testu i soubory deterministických testů mohou být do paměti procesoru nahrány pomocí externího testeru.

Jedním z problémů této metody je generace realizovatelných součástkových testů. Protože zavedení součástkového testu závisí na funkcionalitě obvodu, je nemožné zavést některé testovací sekvence. Proto musí součástkový test splňovat určité podmínky, zavedené v souboru instrukcí daného procesoru, jako platnost vstupních hodnot a závislost mezi vstupy. I když datové vstupy nepodléhají žádným podmínkám, kontrolní vstupy přesto často obsahují množství podmínek, protože jsou tvořeny jinými součástkami.

Fáze aplikace testu zahrnuje zavádění testu součástek a analýzu jejich reakcí na použití procesorových instrukcí. Protože jsou testy součástek vyvíjeny za podmínek určených souborem instrukcí procesoru, bude vždy možné vytvořit zaváděcí program, který může být použit pro aplikaci součástkového testu. Speciální péči je ale třeba věnovat shromažďování reakcí na součástkový test. Jak je zmíněno dříve, datové vstupy a kontrolní vstupy mají rozdílné ovládání, proto musí být při přípravě testu dodrženy podmínky. Podobně, na výstupu má datový výstup a stavový výstup rozdílnou dobu odečítání a měly by být při odečítání reakcí na test zpracovány rozdílně. Stavové výstupy mohou být čteny pouze pomocí speciální sekvence instrukcí.

Obr. 22 stručně zobrazuje metodu self-testu. Pokud je použit otisk self-testu, program na generování testů, zabudovaný v čipu, emuluje generátor pseudonáhodných sekvencí a expanduje otisky do testových sekvencí. Testovací sekvence jsou aplikovány na komponenty pomocí programu za rychlosti procesoru, který také shromažďuje odpovědi na test a ukládá je do paměti. Pokud je to vyžadováno, mohou být rekce na test zkompresovány do reakčních otisků za použití programu na analýzu testových reakcí. Tyto reakce jsou uloženy do paměti a mohou být později odtud staženy a použity k analýze.



Obr.22: Stručné zobrazení self-testu

Zaměřením strukturního testu na potřeby malých komponent má tato metoda pokrytí chyb srovnatelné s deterministickým strukturním testováním. Protože jsou aplikace na testování komponent a shromáždění odpovědí vytvořeny s instrukcemi místo se skenovacími řetězci, nevyžadují žádnou režijní potřebu prostoru nebo výkonu, a aplikace testu je prováděna za chodu. Mnohem důležitější je, že přenesením role externích testerů od aplikací testů a monitorování odpovědí k nahrávání testovacích programů a stahování reakcí, tato metoda umožňuje testování GHz procesorů za chodu s běžnými testery.

7 Aplikace metody FMEA na procesor AVR

Pro ukázku aplikace konkrétní metody a vytvoření testovacího scénáře jsem si zvolil metodu FMEA. Aplikace bude probíhat s použitím aritmeticko-logické jednotky ALU procesoru AVR 8 bit z rodiny procesorů Atmel.

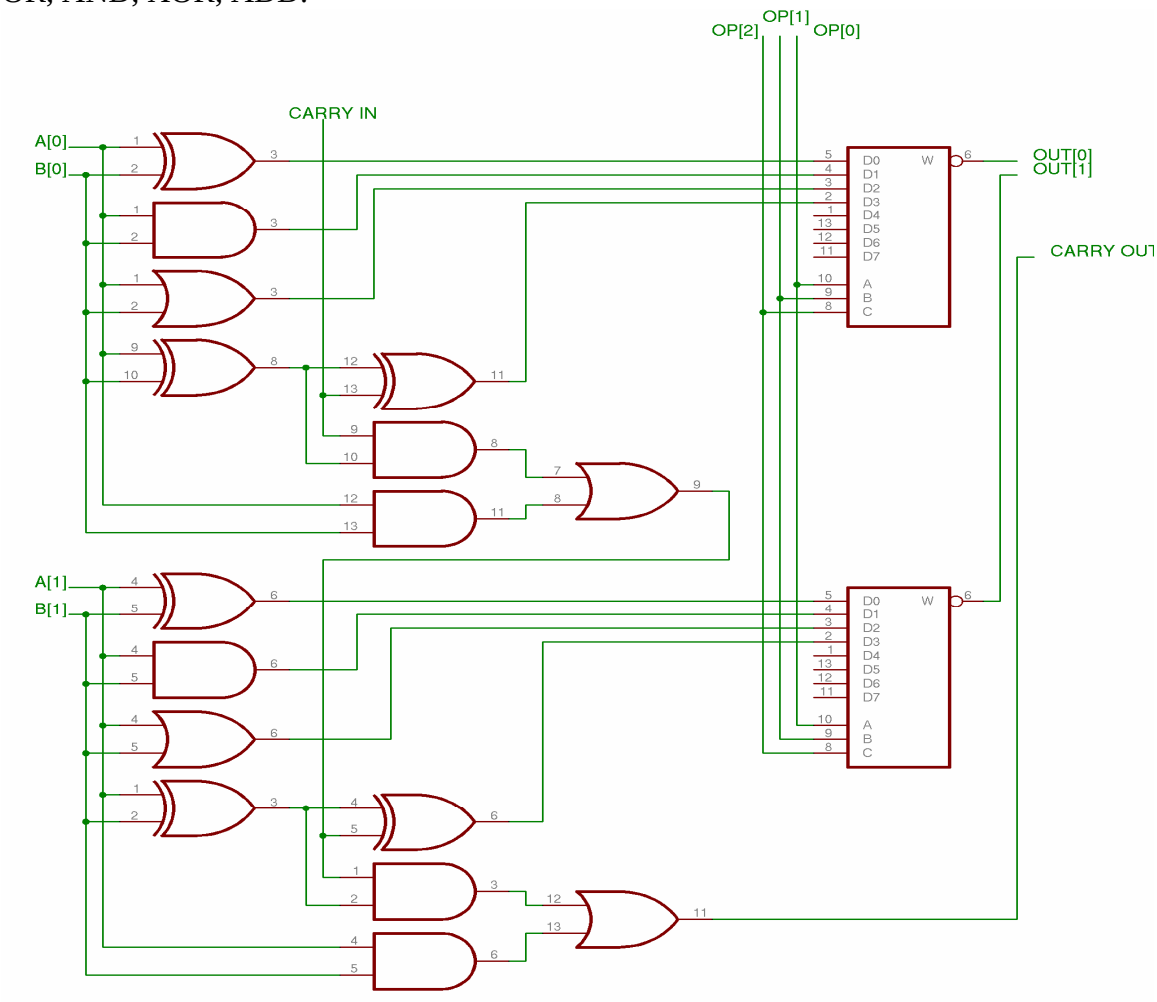
7.1 Design mikroprocesoru – ALU

Mikroprocesory obsahují jeden modul, který provádí aritmetické operace s celými čísly. To proto, že mnoho různých aritmetických a logických operací může být provedeno použitím podobného, či stejného hardwaru. Součástka, která provádí aritmetické a logické operace se nazývá Aritmeticko-logická jednotka – ALU.

ALU je jednou z nejdůležitějších součástí mikroprocesoru, a většinou je to i část procesoru, která je navržena jako první. Jakmile je ALU vytvořena, zbytek mikroprocesoru je implementován tak, aby poskytoval operandy a kontrolní kódy pro ALU.

Jednotky ALU musejí obyčejně být schopné provádět základní logické operace (AND, OR) včetně operace ADD. Zahrnutí invertorů na vstup umožňuje ALU se stejným hardwarem provádět operaci odčítání (přidáním invertovaného operandu), a také operace NAND a NOR.

Na následujícím Obr. 23 [6] je příklad základní 2 bitové jednotky ALU. Jednotky na pravé straně obrázku jsou multiplexery, které se používají pro výběr mezi různými operacemi: OR, AND, XOR, ADD.



Obr. 23: Jednotka ALU pracující se dvěma bity [6]

Tato 2-bitová ALU jednotka má dva vstupy pojmenované A a B: A[0] a B[0] jsou bitem s nejnižším významem a A[1] a B[1] jsou bitem s nejvyšším významem. Každý bit z této ALU je identický s výjimkou zpracování carry bitu.

Zpracování jednoho bitu je následující: A a B vstupy vedou do čtyř hradel nalevo (od shora dolu): XOR, AND, OR, a XOR. Horní tři hradla provádějí operace XOR, AND a OR na vstupu z A a B. Poslední hradlo je počátečním hradlem pro sčítač.

Posledním krokem pro každý bit je multiplexer na konci. Vstup OP o třech bitech z kontrolní jednotky určuje, která z funkcí bude na výstupu.

- OP = 000 → XOR
- OP = 001 → AND
- OP = 010 → OR
- OP = 011 → sčítání

Je jasné že zbylé čtyři vstupy multiplexeru jsou připraveny pro další funkce (odčítání, násobení, dělení, atd.). Ačkoliv není OP[2] v tomto případě použito (i když je zapojeno), bude třeba v případě, že budeme chtít používat více než jen 4 operace naznačené nahoře.

Obvody carry in a carry out jsou obyčejně připojeny k nějaké formě stavového registru.

Je důležité si všimnout, že všechny operace jsou prováděny paralelně a zvolený signál OP je , jak už bylo řečeno, použit k určení, jaký výsledek je poslán dál po datovém vedení. Stejně tak je nutné si všimnout, že carry signál, který je použit jen při operaci sčítání, je generován a poslán do ALU při každé operaci, takže je důležité ho ignorovat, pokud neprovádíme operaci sčítání.

ALU jednotky pracující s více bity jsou podstatě jenom rozšířením této jednotky o další obvody a hradla, a poté dokáží provádět i další operace, jejich vnitřní složitost pak dosahuje vysokých hodnot.

Logické operace a sčítání jsou jedny z nejlehčích, ale také z nejčastějších operací. Z toho důvodu je typická ALU jednotky navržena ke zpracování speciálně těchto operací, a další operace, jako násobení a dělení, jsou zpracovávány v odděleném modulu.

7.1.1 ALU konfigurace

Pro bližší porozumění funkce ALU, a pro aplikace testovacích technik, je nutné pochopit způsob interakce ALU se zbytkem procesoru.

Jakmile je ALU navržena, je nutné definovat, jak reaguje na signály z procesoru. Existuje množství různých konfigurací které můžeme vybrat, každá s přínosy i s problémy. V podstatě jde o to, jak je provedeno načítání a ukládání informací v registrech.

Základní způsoby ukládání do registrů jsou:

Akumulátor

Akumulátor je registr, který ukládá výsledky všech ALU operací, a je také jedním z operandů pro všechny instrukce. To znamená, že sběrnice může být jednodušší, protože instrukce potřebují specifikovat pouze jeden operand, místo obvyklého jednoho operandu a cílového registru. Akumulátorové architektury mají jednoduchou sběrnici a jsou většinou velice rychlé, ale vyžadují dodatečný software pro nahrávání správných hodnot do akumulátoru.

Jedním z příkladů počítačového systému používajícího akumulátor je klasická stolní kalkulačka.

Register-to-register

Jedna z nejběžnějších architektur je Register-to-register architektura. V této konfiguraci může programátor specifikovat oba zdrojové operandy a cílový registr. Bohužel je nutné, aby sběrnice byla rozvržena tak, aby obsahovala prostory jak pro zdrojové operandy tak pro cílové operandy. T vyžaduje větší délku instrukce a také vyžaduje větší úsilí (v porovnání s akumulátorem) pro zápis výsledku zpět do registru po provedení operace. Tento krok zpětného zápisu může způsobovat problémy se synchronizací v některých typech procesorů.

Svazek registrů

Svazek registrů je něco jako kombinace struktur register-to-register a akumulátor. Ve svazku registrů čte ALU operandy z vrcholu svazku a výsledek je zase umístěn na vrcholek svazku. Komplikované matematické operace vyžadují rozklad v jiné formy, což může

programátorům způsobovat problémy. Nicméně mnoho kompilátorů dokáže tento problém snadno vyřešit použitím binárních stromů pro vnitřní reprezentaci instrukcí. Také hardware musí být vytvořen pro implementaci svazku registrů, včetně PUSH a POP operací, které navíc umožní hardwaru detekovat a zvládat chyby ve svazku.

Výhodou je vysoce zjednodušená sběrnice. Operandy nemusejí být specifikovány, protože všechny operace probíhají ve specifických lokacích ve svazku.

Registr s pamětí

Jednou s komplikovanějších struktur je registr s pamětí. V tomto systému je jeden operand načítán z registru a druhý z vnější paměti. V této struktuře je sběrnice složitější, protože každá instrukce musí být schopná uložit kompletní adresu v paměti, která může být velice dlouhá. V praxi není toto schéma přímo použito, ale je většinou integrováno do jiného schématu, například do register-to-register, pro zajištění pružnosti. Některé architektury mají možnost specifikace jednoho z operandů do instrukce jako adresu v paměti, ačkoliv jsou většinou specifikovány jako adresy v registru.

Existuje mnoho dalších komplikovanějších struktur, některé z nich odlišné, některé jsou kombinací struktur popsaných výše. Závisí to v podstatě na návrháři, přesné rozhodnutí o navržení struktury mikroprocesoru a způsobu poskytováni dat pro ALU.

Protože se architektury ALU a způsob práce s registry u jednotlivých modelů liší, musejí být odlišné i testovací schémata pro různá ALU. Při vytváření těchto schémat je nutné zejména vzít v potaz způsob ukládání operandů do registrů, protože při testování funkce všech instrukcí najednou pomocí jednoho řetězce příkazů musí být ošetřeno ukládáni a následné načítání operandů z registrů.

V následující Tab. 6 si nejprve uvedeme přehled použitých aritmeticko-logických instrukcí u procesoru řady Atmel AVR..

Mnemotechnika	Operand	Popis	Operace
ADD	Rd, Rr	Součet bez Carry	Rd ← Rd + Rr
ADC	Rd, Rr	Součet s Carry	Rd ← Rd + Rr + C
ADIW	Rd,K	Přičtení konstanty ke slovu	Rd + 1 : Rd ← Rd + 1 : Rd + K
SUB	Rd,Rr	Odečtení bez Carry	Rd ← Rd - Rr
SUBI	Rd,K	Odečtení konstanty	Rd ← Rd - K
SBC	Rd,Rr	Odečtení s Carry	Rd ← Rd – Rr - C
SBCI	Rd,K	Odečtení konstanty s Carry	Rd ← Rd – K - C
SBIW	Rd,K	Odečtení konstanty od slova	Rd + 1 : Rd ← Rd + 1 : Rd - K
AND	Rd,Rr	Logické AND	Rd ← Rd · Rr
ANDI	Rd,K	Logické AND s konstantou	Rd ← Rd · K
OR	Rd,Rr	Logické OR	Rd ← Rd v Rr
ORI	Rd,K	Logické OR s konstantou	Rd ← Rd v K
XOR	Rd,Rr	Exklusive OR	Rd ← Rd ⊕ Rr
COM	Rd	Doplněk jedné	Rd ← $FF - Rd
NEG	Rd	Doplněk dvou	Rd ←$00 - Rd
SBR	Rd,K	Nastavení bitů v registru	Rd ← Rd v K
CBR	Rd,K	Odstranění bitů z registru	Rd ← Rd · ($FFh – K)
INC	Rd	Přírůstek	Rd ← Rd + 1
DEC	Rd	Úbytek	Rd ← Rd – 1
TST	Rd	Test na nulu nebo na minus	Rd ← Rd · Rd
CLR	Rd	Vyčištění registru	Rd ← Rd ⊕ Rd
SER	Rd	Nastavení registru	Rd ← $FF
MUL	Rd,Rr	Násobení bez znaménka	R1 : R0 ← Rd × Rr
MULS	Rd,Rr	Násobení se znaménkem	R1 : R0 ← Rd × Rr
MULSU	Rd,Rr	Násobení kombinované	R1 : R0 ← Rd × Rr
FMUL	Rd,Rr	Zlomkové násobení bez znaménka	R1 : R0 ← (Rd × Rr) << 1
FMULS	Rd,Rr	Zlomkové násobení se znaménkem	R1 : R0 ← (Rd × Rr) << 1
FMULSU	Rd,Rr	Zlomkové násobení kombinované	R1 : R0 ← (Rd × Rr) << 1

Tab. 6: Souhrn aritmetických a logických instrukcí

V předchozí tabulce jsou obsaženy tyto registry a operandy:

Rd: Cílový (a zdrojový) registr v Souboru registrů
Rr: Zdrojový registr v Souboru registrů
K: Konstanta
C: Carry bit

Pro další analýzu funkce ALU při vykonávání aritmeticko-logických instrukcí je nutné určit rozdíl mezi instrukcemi s Carry bitem a bez Carry bitu.

V počítačových procesorech je carry bitem označován jediný bit v systémovém stavovém registru, používaný k indikaci, zda došlo k přenosu čísla z jednoho sloupce čísel do jiného sloupce více významných čísel během výpočtu algoritmu. Bývá generován z nejvýznačnějšího ALU bitu. Při použití po aritmetické operaci může být považován za neoznačený ekvivalent k bitu přetečení.

V kompilačním jazyku architektury x86 může být carry bit použit k mnoha instrukcím. Ty zahrnují například ADC (Součet s Carry), z instrukcí logického posunu je to: SHL (Logické posunutí vlevo) a SHR (Logické posunutí vpravo) a z instrukcí rotace skrze carry je to: RCR (Rotace skrze Carry vpravo) a RCL (Rotace skrze Carry vlevo). V těchto instrukcích může být carry bit použit i jako vstup (výsledek závisí na hodnotě carry bitu) nebo jako výstup (carry bit se mění podle instrukcí). Použití carry bitu tímto způsobem umožňuje operace sčítání více slov, dělení, posunu nebo rotace.

Pro naše účely postačí vědět, že logické obvody vykonávající instrukce jsou stejné pro instrukci s carry bitem i bez carry bitu, takže nám postačí otestovat pouze jednu z těchto dvojic instrukcí.

7.1.2 Význam jednotlivých instrukcí

ADD – Sčítání bez Carry
Sčítá dva registry bez Carry bitu a umístí výsledek do cílového registru Rd.

ADC – Sčítání s Carry
Sčítá dva registry a Carry bit a umístí výsledek do cílového registru Rd.

ADIW – Přičtení hodnoty ke slovu
Přičte konstantní hodnotu (0 – 63) k registrovému páru a umístí výsledek do registrového páru. Tato instrukce pracuje na horních čtyřech registrových párech a je vhodná pro operace na ukazatelových registrech.

SUB – Odečítání bez Carry
Odečítá mezi dvěma registry a umístí výsledek do cílového registru Rd.

SUBI – Odečítání konstanty
Odečítá mezi registrem a konstantou a umístí výsledek do cílového registru Rd.

SBC – Odečítání s Carry
Odečítá mezi dvěma registry a odečítá s Carry bitem a umístí výsledek do cílového registru Rd.

SBCI – Odečítání konstanty s Carry
Odečítá konstantu od registru a odečítá s Carry bitem a umístí výsledek do cílového registru Rd.

SBIW – Odečítání konstanty od slova
Odečítá konstantní hodnotu (0 – 63) od registrového páru a umístí výsledek do registrového páru. Tato instrukce pracuje na horních čtyřech registrových párech a je vhodná pro operace na ukazatelových registrech.

AND – Logické AND
Provádí operaci logické AND mezi obsahem registru Rd a registru Rr a umístí výsledek do cílového registru Rd.

ANDI – Logické AND s konstantou
Provádí logické AND mezi obsahem registru Rd a konstantou a umístí výsledek do cílového registru Rd.

OR – Logické OR
Provádí operaci logické OR mezi obsahem registru Rd a registru Rr a umístí výsledek do cílového registru Rd.

ORI – Logické OR s konstantou

Provádí logické OR mezi obsahem registru Rd a konstantou a umístí výsledek do cílového registru Rd.

XOR – Exklusive OR

Provádí operaci logické XOR mezi obsahem registru Rd a registru Rr a umístí výsledek do cílového registru Rd.

COM – Doplněk jedné

Tato instrukce provádí funkci Doplněk jedné na registru Rd.

NEG – Doplněk dvou

Přepíše obsah registru Rd s jeho doplňkem dvou.

SBR – Nastavení bitů v registru

Nastaví konkrétní bity v registru Rd. Provádí logické ORI mezi obsahem registru Rd a konstantní hodnotou K a umístí výsledek do cílového registru Rd.

CBR – Odstranění bitů z registru

Odstraní konkrétní bity v registru Rd. Provádí logické AND mezi obsahem registru Rd a doplňkem konstantní hodnoty K. Výsledek umístí do cílového registru Rd.

INC – Přírůstek

Přidá jednu -1- k obsahu registru Rd a umístí výsledek do cílového registru Rd.

DEC – Úbytek

Odečte jednu -1- od obsahu registru Rd a umístí výsledek do cílového registru Rd.

TST – Test na nulu nebo mínus

Testuje, zda je obsahem registru nula nebo negativní hodnota. Provádí logické AND mezi registrem a tím samým registrem. Registr zůstane nezměněn.

CLR – Vymazání registru

Vymaže registr. Tato instrukce provede operaci XOR mezi registrem a tím samým registrem. Tím se smažou všechny bity v registru.

SER – Nastavení všech bitů v registru

Nahraje hodnotu $FF přímo do registru Rd.

MUL – Násobení bez znaménka

Tato instrukce provádí 8-bit × 8-bit → 16-bit násobení záporných hodnot. Operace násobení musí být rozdílná mezi kladnými a zápornými hodnotami, protože jednoduchost Doplňku dvou NEG se nepřenáší na násobení. Instrukce MUL násobí kladné hodnoty.

MULS – Násobení se znaménkem

Tato instrukce provádí 8-bit × 8-bit → 16-bit násobení kladných hodnot. Tato instrukce je pouze pro záporná čísla, a netvoří negativní hodnoty výsledku.

MULSU – Násobení kladných s zápornými

Tato instrukce provádí 8-bit × 8-bit → 16-bit násobení kladných a záporných čísel.

FMUL – Zlomkové násobení bez znaménka

Tato instrukce provádí 8-bit × 8-bit → 16-bit násobení záporných hodnot a posune výsledek o jeden bit vlevo.

FMULS – Zlomkové násobení se znaménkem

Tato instrukce provádí 8-bit × 8-bit → 16-bit násobení kladných hodnot a posune výsledek o jeden bit vlevo.

FMULSU – Zlomkové násobení kladných s zápornými

Tato instrukce provádí 8-bit × 8-bit → 16-bit násobení kladných a záporných hodnot a posune výsledek o jeden bit vlevo.

7.2 Možnosti testování instrukcí

Způsobů jak otestovat správnou funkci aritmeticko logických instrukcí v ALU je několik. V podstatě nejjednodušším způsobem je otestování každé z instrukcí zvlášť, kdy zadáme hodnoty do vstupních registrů a porovnáme hodnotu výstupního registru s očekávanou hodnotou. Výhodou tohoto způsobu je jednoduchost a rychlost otestování jednotlivých instrukcí, nevýhodami pak otestování pouze proti jednomu druhu chyb a při větším množství instrukcí i zdlouhavé.

7.2.1 Příklady testování jednotlivých instrukcí

Jednoduše otestovat základní instrukce ADD, SUB, AND, OR, XOR z předchozí tabulky lze například takto:

Jako vstupní hodnoty použijeme např. čísla 10 a 20, což je v binární soustavě 1010 a 10100.

Funkce ADD provádí operaci Rd ← Rd + Rr, takže po vyplnění vstupních registrů čísly 1010 a 10100 by se měla ve výstupním registru objevit hodnota 11110.

Funkce SUB provádí operaci Rd ← Rd – Rr, takže po vyplnění vstupních registrů čísly 10100 a 1010 by se měla ve výstupním registru objevit hodnota 1010.

Funkce AND provádí operaci Rd ← Rd · Rr, neboli logické AND, kdy je na výstupu 1 pouze pokud je na všech vstupech 1, takže při vyplnění vstupních registrů čísly 1010 a 10100 by se ve výstupním registru měla objevit hodnota 10100.

Funkce OR provádí operaci Rd ← Rd v Rr, neboli logické OR, kdy je na výstupu 1 vždy, když je alespoň na jednom vstupu 1, takže po vyplnění vstupních registrů čísly 1010 a 10100 by se ve výstupním registru měla objevit hodnota 10100.

Funkce XOR provádí operaci Rd ← Rd ⊕ Rr, neboli logické Exklusive-OR, kdy je na výstupu 1 vždy, když jsou vstupy rozdílné, takže po vyplnění vstupních registrů čísly 1010 a 10100 by se ve výstupním registru měla objevit hodnota 00001.

Při takovém to testování jednotlivých instrukcí ovšem narazíme na řadu problémů. První z nejasností nastane při použití binárních čísel o různých délkách, kdy po zkombinování hodnot o délce prvního čísla je nejasné, které hodnoty budou na prázdných místech a budou se kombinovat s hodnotami na posledních pozicích delšího čísla. Proto je nutné jasně definovat hodnoty volných pozic v registrech, pro toto testování předpokládám že volná místa mají hodnotu 0.

Další z problémů je zřejmý na testování funkcí AND a OR, kdy nám vyjde stejný výsledek, v konkrétním případě sice správný, ale díky tomu nejsme schopni rozlišit o jakou funkci se jedná. Může jít totiž o správnou činnost každé z funkcí, nebo o chybu, kdy jedno z hradel pracuje chybně nebo dokonce vykonává jinou funkci.

Testování správné funkce hradel není možné pouze provedením jednoho výpočtu, jak ukazuje předchozí příklad. V podstatě není možné otestovat správnou funkčnost na 100%.

7.2.2 Testovací algoritmus a otestování celého setu instrukcí

Pro otestování celého setu aritmetickologických instrukcí je třeba navrhnout vhodný algoritmus, který po zadání vstupních dat provede otestování všech funkcí. Je možné vytvořit algoritmus, který provede testování každé funkce zvlášť, to je ale nepraktické při velkém počtu funkcí, kdy je třeba mít pro každou z funkcí připravena vstupní data a hlavně následná kontrola výstupních hodnot může být velice zdlouhavá. Elegantnější je proto řešení, kdy algoritmus použije vstupní data pouze na první funkci a výstupní data použije následně pro otestování další funkce. Data, která dostaneme na výstupu pak pouze porovnáme s předpokládanou hodnotou. Je ovšem nutné zajistit, aby v průběhu programu byl po každé instrukci proveden záznam aktuálních hodnot registrů pro případ, že výsledná data nebudou souhlasit s předpokládanými. Pak lze snadno dohledat, která z instrukcí způsobila chybu. Takový algoritmus musí být ale navržen pečlivě, velice důležitá je volba vstupních dat.

Pro vytvoření testovacího algoritmu se použijí instrukce popsané výše a zdrojový kód se vytvoří pomocí příkazů uvedených v [3]. Při tvorbě testovacího algoritmu lze otestovat pouze jednou instrukci, která se vyskytuje dvakrát, jednou samotná a podruhé s Carry. To z toho důvodu, že obě instrukce, bez Carry i s Carry, vykonává stejný obvod, jen je navíc zapojen Carry bit.

Použití jednotlivých příkazů:

```
Adc:                        ; Přičte R1:R0 do R3:R2
                add   r2, r0    ; Přičte nízký byte
                adc   r3, r1    ; Přičte vysoký byte s carry
Adiw:           adiw  r25:24, 1 ; Přičte 1 do r25:r24
Sbc:                        ; Odečte r1:r0 od r3:r2
                sub   r2, r0    ;Odečte nízký byte
                sbc   r3, r1    ;Odečte vysoký byte s carry
Sbiw:           sbiw   r25:r24, 1 ;Odečte 1 od r25:r24
And:            and   r2, r3    ;Bitový and mezi r2 a r3, výsledek v r2
Andi:           andi  r16, $0F  ;Bitový and mezi r16 a hodnotou $0F, nastavení
                                ;nižší čtveřice bitů v registru
Or:             or    r15, r16  ;Bitové or mezi registry
Ori :           ori   r16, $F0  ;Bitové or mezi r16 a hodnotou $F0, nastavení
                                ;vyšší čtveřice bitů v registru
Eor:            eor   r4, r4    ;Vyčistí r4
                eor   r0, r22   ;Bitové exclusive or mezi r0 a r22
Com:            com   r4        ;Vytvoří doplněk jedné z r4 a nahradí stávající
                                ;hodnotu
Neg:            neg   r11       ;Vytvoří doplněk dvou z r11 a nahradí stávající
                                ;hodnotu
Sbr:            sbr   r16, 3    ;Nastaví bity 0 a 1 v r16
                sbr   r17, $F0  ;Nastaví 4 nevíce významné bity v r17
Cbr:            cbr   r16, $F0  ;Smaže vyšší čtveřici bitů v r16
                cbr   r18, 1    ;Smaže bit 0 v r18
Inc:            inc   r22       ;Zvýší hodnotu r22 o 1
Dec:            dec   r17       ;Sníží hodnotu r17 o 1
Tst:    tst     r0              ;Test r0
Clr:            clr   r17       ;Smaže r17
Ser:            ser   r17       ;Nahraje hodnotu $FF přímo do registru
Mul:            mul   r5, r4    ;Znásobí r5 a r4
Muls:           muls  r22, r20  ;Znásobí r22 a r20
```

Algoritmus schopný prověřit všechny tyto instrukce by mohl být následující:

- použijeme registry r0 až r4, testovací sekvence bude nahraná v registrech r0, r1, r2, r3 a konstantní hodnota k v externí paměti
- vstupní proměnné jsou hodnoty v registrech r0 až r3, dále konstanty C pro carry a k.

- při použití počátečních hodnot r0:11111111, r1: 10101010, r2:11110000, r3: 00001111, k:10.

Operační algoritmus		Aktuální hodnoty registrů				Využívané součásti obvodu							
Funkce	Operandy	r0	r1	r2	r3	AND	OR	XOR	ADD	SUB	CARRY	reg-to-reg	reg-to-mem
add	r2, r0	11111111	10101010	11101111	00001111				x			x	
adc	r3, r1	11111111	10101010	11101111	10111001				x		x	x	
adiw	r2, r3, k	11111111	10101010	11110001	10111011				x			x	x
sub	r0, r2	00001110	10101010	11110001	10111011					x		x	
sbc	r3, r1	00001110	10101010	11110001	00010001					x	x	x	
sbiw	r2, r3, k	00001110	10101010	11101111	00001111					x		x	x
and	r2, r3	00001110	10101010	00000111	00001111	x						x	
andi	r0, $0F	00001110	10101010	00000111	00001111	x							x
or	r1, r0	10101110	10101010	00000111	00001111		x					x	
ori	r0, $F0	11111110	10101010	00000111	00001111		x						x
eor	r2, r3	11111110	10101010	00001000	00001111			x				x	
com	r1	11111110	01010101	00001000	00001111					x			x
neg	r0	00000010	01010101	00001000	00001111					x			x
sbr	r0, 3	00000111	01010101	00001000	00001111		x						x
cbr	r3, 1	00000111	01010101	00001000	00000111	x							x
inc	r0	00001000	01010101	00001000	00000111				x				x
dec	r1	00001000	01010100	00001000	00000111					x			x
tst	r0	00001000	01010100	00001000	00000111	x						x	
clr	r2	00001000	01010100	00000000	00000111			x				x	
ser	r0	11111111	01010100	00000000	00000111								x

Tab. 7: Určení duplexních funkcí v algoritmu

Při návrhu testovacího algoritmu je nutné pokusit se o odstranění duplexních funkcí. Některé z instrukcí jsou jen modifikací jiných obecnějších instrukcí. Jak zobrazuje Tabulka 7, část aritmeticko-logických instrukcí využívá stejné části ALU jako jiné instrukce. Proto je vhodné vybrat ty, které obsáhnou největší část z hardwarové výbavy ALU. Otestováním jedné obecnější instrukce obsáhneme obvody používané jinými, více specificky zaměřenými instrukcemi. Díky tomu můžeme testovací algoritmus zmenšit při zachování stejné míry pokrytí chyb, a tím i snížit nároky na výpočetní výkon a čas.

Po odstranění duplicitních instrukcí je algoritmus doplněn funkcí CP (Compare), která po každém kroku porovná aktuální hodnoty registrů s očekávanými. V případě nesouhlasu se běh programu pomocí funkce BRNE (Branch if not equal) přeruší a je nahlášena chyba. Je nutné přidat externí paměť m0, m1, m2, m3 obsahující očekávané hodnoty registrů pro porovnání funkcí CP. Paměť může obsahovat pouze aktuální hodnotu, která se po správném provedení

každé instrukce přepíše, nebo může obsahovat všechny hodnoty registrů pro celý průběh programu, a funkce jen odkazuje na aktuální místo v paměti.

Testovací algoritmus je tedy po odstranění duplexních instrukcí a přidání srovnávací funkce následující:

```
adc   r3, r1          ; přičte obsah r1 do r3
cpc   r3, m3          ; porovná obsah r3 s m3
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
adiw  r2, r3, k       ; přičte hodnotu k do r2 a r3
cp    r2, m2          ; porovná obsah r2 a m2
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
cp    r3, m3          ; porovná obsah r3 a m3
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
sbc   r3, r1          ; odečte obsah r1 od r3
cpc   r3, m3          ; porovná obsah r3 a m3
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
sbiw  r2, r3, k       ; odečte hodnotu k od r2 a r3
cp    r2, m2          ; porovná obsah r2 a m2
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
cp    r3, m3          ; porovná obsah r3 a m3
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
and   r2, r3          ; provede logické AND mezi r2 a r3
cp    r2, m2          ; porovná obsah r2 a m2
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
andi  r0, $0F         ; provede logické AND mezi obsahem r0 a hodnotou $0F
cp    r0, m0          ; porovná obsah r0 a m0
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
or    r1, r0          ; provede logické OR mezi r1 a r0
cp    r1, m1          ; porovná obsah r1 a m1
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
ori   r0, $F0         ; provede logické OR mezi obsahem r0 a hodnotou $F0
cp    r0, m0          ; porovná obsah r0 a m0
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
eor   r2, r3          ; provede logické XOR mezi r2 a r3
cp    r2, m2          ; porovná obsah r2 a m2
brne  error           ; přeruší běh programu pokud hodnoty nesouhlasí
```

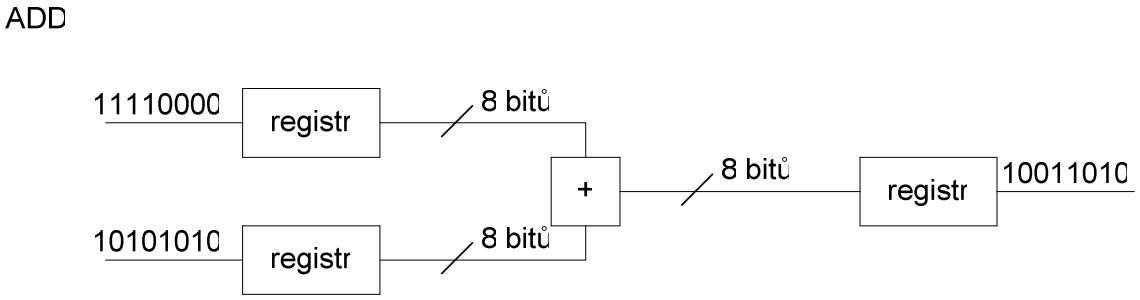
Tento algoritmus obsahující reprezentativní zástupce všech aritmeticko logických instrukcí může být dále upraven vhodným uspořádáním posloupnosti vykonávání příkazů.

7.3 Aplikace metody FMEA na hradlo

Důvodů, proč mohou jednotlivá hradla selhat je celá řada, proto je vhodné provést analýzu FMEA na každé hradlo. V následujícím příkladu je provedena analýza FMEA na hradle ADD. Aplikace této metody je u ostatních hradel obdobná.

Jak je zobrazeno na obr. 24, hradlo ADD obsahuje dva vstupy a jeden výstup. Vstupní hodnoty jsou načítány z registrů a výsledná hodnoty je opět uložena do registru. Vstupní i výstupní registry jsou 8 bitové. Každý z bitů může při chybě nabývat tři hodnoty: hodnotu 0,

hodnotu 1 a invertovanou hodnotu ke správné hodnotě. V každém registru tedy může při osmi bitových pozicích být 24 stavů, z čehož 16 je chybných. Daný bit může být nastaven správně, nebo může být vždy nastaven na 1 nebo na 0, a nebo může být vždy invertován.



Obr. 24: Model funkce ADD

Ke špatnému nastavení bitu může dojít ať už ve vstupních či výstupních registrech, na vstupech a výstupech hradla či v samotném hradlu. Při posuzování následků chyby na systém nás zajímá především správnost bitů na ve výstupním registru. Při počtu 24 možných stavů na každém vstupu a výstupu dostaneme celkem 72 možných stavů jednotlivých bitů. Z toho správných stavů je pouze 24. Na výstupním registru je z 24 celkových stavů pouze 8 správných, takže dvě třetiny možných chybných výstupních stavů je nutné pokrýt při chybové analýze.

V tabulce 8 je vytvořena jednoduchá FMEA analýza tohoto hradla. Jsou uvažovány možné příčiny změn hodnot v jednotlivých bitových pozicích. Je zde uvažován vliv okolního prostředí na vstupní a výstupní obvody, což může způsobit dané chyby. Dále jsou určena čísla kritičnosti, četnosti výskytu a detekce. Výše těchto údajů je určena pouze odhadem, protože jak je popsáno výše, tyto údaje jsou získávány především ze zkušeností z podobných systémů, které v tomto případě nebyli k dispozici. Všechny tyto čísla nabývají hodnot od 1 do 10, v případě kritičnosti a četnosti výskytu vzestupně, u detekce potom sestupně. Výsledná hodnota čísla priority rizika se učí vynásobením předchozích čísel. Tento údaj slouží k určení závažnosti dané chyby a určuje pořadí nápravných opatření.

Obdobná analýza se provede s každou aritmeticko logickou instrukcí.

Jednotka/ Funkce	Potenciální způsob selhání	Potenciální efekt selhání	Kritičnost	Potenciální příčina/ mechanismus selhání	Četnost výskytu	Kontrola systému	Detekce	Č.P.R
Hradlo ADD	Zkrat mezi vstupy	Nesprávní funkce hradla - bude fungovat správně pouze když na obou vstupech bude 0	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	5	Testování při výrobě, znalosti z minulosti	8	280
	Trvalé napojení vstupu na 0 – zkrat na uzemnění	Nesprávná funkce hradla, je pouze přenesena hodnota druhého vstupu	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	5	Testování při výrobě, znalosti z minulosti	6	210
	Trvalé napojení výstupu na 0 – zkrat na uzemnění	Nesprávná funkce hradla, výstup je vždy 0	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	5	Testování při výrobě, znalosti z minulosti	2	70
	Trvalé napojení vstupu na 1 – zkrat na napájení	Nesprávná funkce hradla, výstupní hodnota je chybná	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	3	Testování při výrobě, znalosti z minulosti	6	126
	Trvalé napojení výstupu na 1 – zkrat na napájení	Nesprávná funkce hradla, výstup je vždy 1	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	3	Testování při výrobě, znalosti z minulosti	2	42
	Vytvoření negace na vstupu	Nesprávná funkce hradla, výstupní hodnota je chybná	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	1	Testování při výrobě, znalosti z minulosti	8	56
	Vytvoření negace na výstupu	Nesprávná funkce hradla, výstupní hodnota je opačná hodnotě předpokládané	7	Špatné zapojení nebo montáž, přítomnost vody či nečistot v obvodu	1	Testování při výrobě, znalosti z minulosti	5	35
	Náhodné tvoření 0 nebo 1 na vstupu nebo výstupu	Nesprávná funkce hradla, výstupní hodnota je chybná	7	Přítomnost vody nebo nečistot, uvolněné kontakty	2	Testování při výrobě, znalosti z minulosti	9	126
	Hradlo nefunguje	Hradlo nepřenáší hodnoty na výstup	5	Nezapojený výstup, přerušené vnitřní obvody	1	Testování při výrobě, znalosti z minulosti	1	5

Tab. 8: Analýza ADD hradla pomocí FMEA

Závěr

V této práci jsem se věnoval analýze možných poruch vznikajících v mikrokontrolérech a počítačových systémech všeobecně. Zjistil jsem, že může dojít k selhání systému způsobeným mnoha různými příčinami. Základní příčinou je vznik chyby v systému. Tyto chyby mohou být mnoha různých druhů a trvání. Proto bylo nutno vyvinout mnoho technologií pro detekci všech možných druhů chyb.

Tyto metody se dělí na hardwarové a softwarové metody detekce chyb. Aby bylo možno danou chybu odhalit, je nutno znát její účinky na chod systému. Proto vznikají takzvané chybové modely, které popisují chování každého druhu chyb. Všeobecně lze chyby rozdělit na náhodné chyby vznikající vlivem okolního prostředí a chyby systematické, které v systému přetrvávají a bývají zahrnuty v systému samotném.

Další část práce se zabývá možnostmi odvrácení a předcházení těchto chyb, a především technikami tolerance chyb, které využívají redundanci k zajištění spolehlivého chodu systému. Redundance, neboli použití více modulů pro zpracování stejného signálu, pracuje buď na principu porovnávání signálu z více modulů a výběru správného signálu, nebo na principu odstavení chybného modulu a přepnutí na jiný záložní modul, tak aby mohl být chybný modul vyměněn. Tolerance chyb se opět dělí na softwarovou toleranci a hardwarovou toleranci.

Důležitou částí vývoje nového systému je analýza možných chyb, které mohou ovlivnit činnost takového systému. Analytických technik je mnoho druhů, každá používá jiný přístup při řešení problému. V podstatě se snaží popsat všechny možné události, které mohou ovlivnit chod systému a tím způsobit například ohrožení majetku nebo lidských životů. Většinou se jedná o chyby v systému a pomocí některé z analytických technik se snažíme určit pravděpodobnost vzniku takovéto chyby. Nejpoužívanějšími technikami je FMEA a HAZOP. První z nich hledá a analyzuje možné chyby a každou z nich klasifikuje třemi různými parametry, čím určí i závažnost této chyby. Druhá technika slouží k analýze spojení mezi jednotlivými součástkami v systému a tím i možnými následky chyby na celý systém. K tomu využívá seznam pomocných slov.

V další části práce jsou popsané techniky testování procesorů. Každý procesor je nutné otestovat k ověření správné funkčnosti a pro zjištění možné přítomnosti chyb. Žádná z testovacích technik nemá stoprocentní pokrytí chyb, jejich výsledky jsou ale užitečné pro některou z chybových analýz. Nejpoužívanější z technik testování procesorů ale i jiných logických obvodu je ATPG, používající techniku náhodného generování testovacích schémat a tím dosahuje vysoké míry pokrytí chyb.

Konečnou částí je pokus o otestování aritmeticko-logických instrukcí v jednotce ALU použité v procesoru AVR z rodiny Atmel. Nejprve je nutné popsat k čemu jednotky ALU v procesoru slouží, jedná se vlastně o jednu z nejdůležitějších jednotek procesoru. Aritmeticko-logické funkce provádějí základní operace sčítání a odčítání a logické funkce AND, OR, XOR. Seznam aritmeticko-logických operací je delší, ale jak je ukázáno v této práci, většina z nich je pouze kombinací základních funkcí. Proto lze testovací algoritmus zmenšit na otestování těchto základních funkcí a tím vlastně otestovat celý soupis instrukcí.

Poté je provedena analýza FMEA jednotlivých hradel, určující možné chyby ovlivňující činnost těchto hradel. Testování aritmeticko-logických instrukcí je pouze základním otestováním jednotky ALU, určující pouze to, že v době testování prováděla jednotky tyto instrukce správně. Ovšem během provozu mikroprocesoru může dojít k mnoha událostem ovlivňujícím jeho běh a i běh ALU. Proto se provádí analýza FMEA, zkoumající možnosti selhání jednotlivých instrukcí a vliv těchto selhání na činnost celé jednotky. Analýza FMEA také ohodnotí jednotlivé potenciální možnosti selhání a tím určí prioritu s jakou mají být tyto chyby vyřešeny.

Z výše uvedeného vyplívá fakt aplikovatelný na všech systémech, a to že nelze otestovat systém na sto procent, nelze pokrýt všechny chyby v systému, pouze určit pravděpodobnost, s jakou byli chyb pokryty.

Dnešní systémy založené na mikrokontrolérech jsou velice složité a zároveň nezbytně důležité pro funkci většiny elektronických systémů a jsou kladeny vysoké nároky na jejich bezchybnou funkci a proto bylo vyvinuto množství technik zaručující s vyšší či nižší úspěšností dodržení bezpečného provozu aplikací využívajících mikrokontroléry.

Seznam literatury a použitých zdrojů:

[1] Neumann, P.G.: Computer related risks. ACM Press. New York. 1995

[2] Storey, T.: Safety – critical computer systems. Addison Wesley. Essen. 1996

[3] ATMEL, 8-bit AVR Instruction Set, Rev. 0856E – AVR – 11/05

[4] Chen, L., Dey, S.:A deterministic functional self-test methodology for processors. Department of Electrical and Computer Engineering, University of California, San Diego 2005

[5] Bushnell, M. L., Agrawal, V. D.: Essentials of electronic testing. Springer. New York. 2000

[6] www.wikipedia.org

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma
"....."
jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne ……..… ……..……..

(podpis autora)